INTELLIGENCE SURVEILLANCE AND RECONNAISSANCE ASSET ASSIGNMENT FOR OPTIMAL MISSION EFFECTIVENESS

THESIS

Ryan D. Kappedal, Captain, USAF

AFIT/GOR/ENS/08-10

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GOR/ENS/08-10

INTELLIGENCE SURVEILLANCE AND RECONNAISSANCE ASSET ASSIGNMENT FOR OPTIMAL MISSION EFFECTIVENESS

THESIS

Ryan D. Kappedal
Captain, USAF

AFIT/GOR/ENS/08-10

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense or the United States Government.

AFIT/GOR/ENS/08-10

INTELLIGENCE SURVEILLANCE AND RECONNAISSANCE ASSET ASSIGNMENT FOR OPTIMAL MISSION EFFECTIVENESS

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

Ryan D. Kappedal, B.S.

Captain, USAF

March 2008

AFIT/GOR/ENS/08-10

INTELLIGENCE SURVEILLANCE AND RECONNAISSANCE ASSET ASSIGNMENT FOR OPTIMAL MISSION EFFECTIVENESS

Ryan D. Kappedal, B.S.

Captain, USAF

Approved:

Maj. August G. Roesener, Ph.D. Chairman	Date
Maj. Shane N. Hall, Ph.D. Member	Date

AFIT/GOR/ENS/08-10

Abstract

This research develops mathematical programming techniques to solve an intelligence, surveillance, and reconnaissance sensor assignment problem for USSTRATCOM. The problem as specified is hypothesized to be difficult (i.e. NP-HARD). With the smallest test cases, the true optimal solution is found using simple optimization techniques, but, due to intractability, the optimal solutions for larger test cases are not found using these same techniques. Instead, heuristic techniques are applied to several test cases in order to determine the best, robust methodologies to find true or near optimal solutions. Specifically, simulated annealing (SA) is tested for convergence properties across several different parameter settings. This research also utilizes local search techniques with simple exchange neighborhoods of various sizes. Mission prioritization is also examined via a weighted sum scalarization technique.

Dedication

I dedicate this thesis to my wife. Her support, understanding, wonderful cooking, and love made all of this possible.

Acknowledgements

I would like to thank the members of my thesis committee, Maj August Roesener and Maj Shane Hall. Their support, encouragement, knowledge and friendship have been invaluable to me throughout this process. Special thanks to the LaTeX gang for all their help with writing, formatting, and keeping me sane during those many late nights in the lab.

Ryan D. Kappedal

Table of Contents

List of Figures

List of Tables

INTELLIGENCE SURVEILLANCE AND RECONNAISSANCE ASSET ASSIGNMENT FOR OPTIMAL MISSION EFFECTIVENESS

1. Introduction

1.1 Background

1.1.1 Summary of Current Process

With the advent of complex sensor networks to track and detect multiple types of targets with differing priorities, a dedicated methodology for assigning sensors to missions in an operationally-relevant timeframe is required. Of particular note, this problem is of interest to members of USSTRATCOM, as they attempt to improve upon the scheduling of their sensor networks. This topic warrants investigative research due to the inadequacies in the current process. As a high priority mission becomes available, the current process involves cancelling all other missions and tasking all assets against the high priority mission, even though the marginal contribution of some of those assets is very small. Total mission probabilities are currently calculated using an assumption of statistical independence between mission steps. This assumption was made for simplicity sake, but may not always reflect reality. There is currently no methodology available to select an optimal mix of assets to accomplish all missions with a reasonable probability of success.

1.1.2 Process Example

A notional mission is depicted on Figure 1.1. In this example, a target detection prompts a mission. The mission involves four sequential tasks: identification, tracking, queueing, and launch. Each of these tasks requires sensor support in order

to be successful, and each of the available sensors has a given probability of success at the individual stages. However, since the target's position can never be known a priori, and a sensor's capability against a target is largely dependant upon sensor orientation and distance, the probability of success is not constant. Hence, the problem of sensor selection is one that has to be quickly resolved on a routine basis.

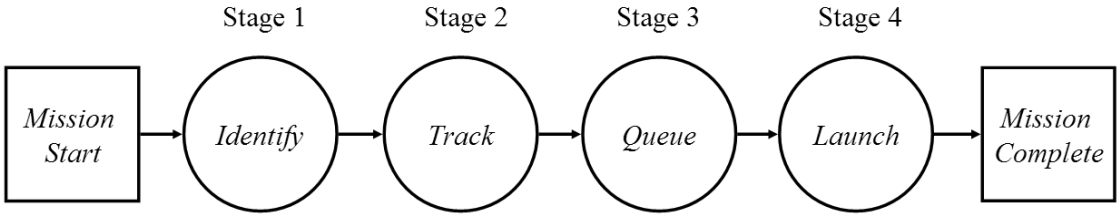

Figure 1.1 Theoretical Mission Stages

1.2 Problem Statement

Given a set of assignable sensors, each of which has a probability of accomplishing a set of missions (each with involves stages), find the binary assignment of sensors to missions (and thereby also to stages) that maximizes the multicriteria, p-dimensional objective function, $F(\mathbf{x})$, where $\mathbf{x}$ is a valid set of assignments. The mathematical problem statement is:

$$\begin{aligned} &\max_{\mathbf{x}\in\Omega} F(\mathbf{x}), \\ &\mathbf{x} \in \mathbb{B}^n \\ &F(\mathbf{x}) \in \mathbb{R}^p \end{aligned} \tag{1.1}$$

where $F(\mathbf{x}) : \mathbb{B}^n \rightarrow \mathbb{R}^p$, n corresponds to the number of binary variables associated with the possible assignments of sensors to missions, and p corresponds to the total number of missions over which we are trying to maximize the probability. Additionally, in the above equation, $\mathbf{x} \in \Omega$ is the set of all feasible sensor assignments. The

specific feasibility constraints (sensors cannot be assigned to more than one mission, etc.) will be discussed in Chapter 3.

1.3 Research Questions

This research seeks to answer the following questions. What multicriteria objective function can model USSTRATCOM's mission success priorities? Can the previously described multicriteria optimization problem for sensor assignment be proven as either a difficult (i.e. NP-COMPLETE) or easy (i.e. P) problem? What solution methodologies (to include heuristic techniques if necessary) could best be used to solve this problem in a timely fashion?

1.4 Research Focus

The main focus of this research will be developing a model that accurately represents the USSTRATCOM sensor assignment problem and solves notional problems of varying size and complexity in a timely fashion. Since probabilities, missions, sensor availabilities, and targets are changing on a daily basis, a robust methodology is preferred.

1.5 Methodology

The development of a robust and efficient solution methodology initially involves constructing a mathematical programming model for all potential missions. In this case, the objective function is a probability of mission success (or some type of expectation) which would be non-linear in nature, making this a Non-Linear Programming (NLP) problem. Some of these probabilities are dynamic and change based on scenario; hence, creating a robust model of analysis is crucial. This problem is hypothesized to be NP-HARD, and heuristic solution methodologies will be

explored to search for better sensor configurations within a reasonable amount of time.

1.6 Assumptions and Limitations

The current assumption for the evaluation of the objective function, $F(x)$, is that the probabilities of all events within the system are independent. USSTRATCOM is sponsoring a separate research effort to conduct the conditional probability computations for $F(x)$. Therefore, that portion of the problem is beyond the scope of this research; the independence assumption will be used in this research.

Another major assumption is that the probability of success for a sensor assigned to a mission's stage is known in sufficient time to be utilized as input. The methodology for calculating these probabilities in an actual situation is largely dependant upon sensor configuration, sensor location, and sensor capability, as well as target location and target type. In some cases, a precise probability of success is not known and estimates are used. For this research, notional probability values will be randomly generated for various scenarios, since the methodology for finding an ideal sensor configuration for any given scenario is of prime interest.

One requirement for this research is a methodology that solves the problem in a reasonable amount of time. Finding the optimal solution to the problem within this time limitation may not always be possible. In this case, heuristic search methods may prove to be preferable to suit USSTRATCOM's needs of quickly finding a high quality solution.

1.7 Implications

The desired result of this research is an efficient methodology for assigning sensors for daily mission assignments in a non-arbitrary, non-biased fashion. This will allow for a greater mission success rate, which could significantly improve the

US Intelligence, Surveillance, and Reconnaissance capabilities. Follow on efforts could be expanded to impact future planning for expanding sensor networks based on potential coverage gaps identified by this research.

1.8 Preview

Chapter 2 outlines the current literature in areas relevant to this research, including: non-linear multi-criteria optimization problems, reliability engineering, and heuristic techniques. Chapter 3 develops a methodology for sensor assignment based on current non-linear multi-criteria optimization problems. Chapter 4 describes solution methods for the model outlined in Chapter 3, by applying the model to problem instances similar to USSTRATCOMs configurations. Chapter 5 completes this research with conclusions and recommendations for future research.

2. Literature Review

This chapter discusses the literature pertinent to the assignment of Intelligence, Surveillance and Reconnaissance (ISR) sensors to accomplish various missions.

The first section outlines basic complexity classes and problem reduction. The second section describes models which relate to the techniques that will be developed in this research. The third section examines sensor assignment methodologies currently used to create redundant, reliable networks. The fourth section discusses techniques for comparing feasible solutions to multicriteria optimization problems (MOP) in order to determine which solutions are better. This chapter concludes by outlining heuristics techniques that are applicable to the model developed in this research.

2.1 Complexity

Prior to presenting background information specifically for this problem, it is necessary to include a brief, general discussion of problem computational complexity.

2.1.1 Decision Problems

Problem classification theory does not directly address optimization problems, but rather applies to *decision problems*. A *decision problem* refers to a problem that has a specific YES-NO answer. In the case of the previously discussed optimization problem given by: $\max \{F(\mathbf{x}) : \mathbf{x} \in \Omega, \mathbf{x} \in \mathbb{B}^n\}$, the corresponding decision problem is stated as: Does there exist an $\mathbf{x} \in \Omega$ with value $F(\mathbf{x}) \geq k$ [40]?

2.1.2 Classes P, NP and NP-Complete

The classes NP and P refer to the number of elementary mathematical operations (i.e., additions, multiplications, comparisons, etc.) that are required to solve

a given problem as a function of the problem's size. For any problem in NP, given a feasible answer to the decision problem (a YES), the fact that the answer is YES can be verified in a polynomial number of operations or *polynomial time*. NP signifies non-deterministic, polynomial time and P signifies polynomial time, hence P is a subset of NP ($P \in NP$). Given a decision problem $Q \in P$, suppose an algorithm exists that solves Q in polynomial time; this implies that Q is *easy*. In contrast, problems for which there has *not yet* been found an algorithm to solve in polynomial time are said to be in a class called NP-COMPLETE. The class NP-COMPLETE was proven to be non-empty using the SATISFIABILITY problem [8]. Under the working hypothesis that $P \neq NP$ (implying that a polynomial time algorithm to solve all problems in NP will *never* be found), the respective classes are shown in Figure 2.1 [29].

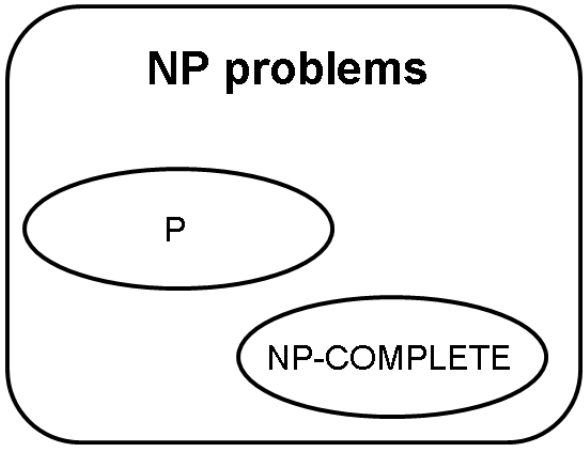

Figure 2.1 P, NP, and NP-COMPLETE, if $P \neq NP$

2.2 Review of Well-known Problems

The next step involves reviewing some well-known optimization problems that pertain to the approach of this research. First, the 0-1 knapsack problem (KP) and its variants (to include the simple allocation problem (SAP), both linear and non-linear) will be examined.

2.2.1 0-1 Knapsack Problem

The 0-1 knapsack problem (KP) is easily described [40]. Imagine that a hiker is preparing for an overnight hike and can only carry a knapsack with weight capacity b, consisting of n possible items, where a_j is the weight of item j, and c_j is the return or "utility" gained by carrying item j. The goal of the problem is to maximize the overall utility of the knapsack, while never exceeding the weight constraint. A decision variable, x_j, has the value of 1 if the item is put in the knapsack and 0 if the item is left behind. KP has the following mathematical form:

$$\max \sum_{j=1}^{n} c_j x_j, \tag{2.1}$$

subject to

$$\sum_{j=1}^{n} a_j x_j \leq b$$
$$\mathbf{x} \in \mathbb{B}^n$$

Its corresponding decision problem has been shown to be NP-COMPLETE [25].

The original knapsack problem as formulated, however, has a much simpler formulation: given a_j, and b, find the binary assignment such that:

$$\sum_{j=1}^{n} a_j x_j = b \tag{2.2}$$
$$\mathbf{x} \in \mathbb{B}^n$$

It is this simplified version of knapsack upon which the original NP-COMPLETE proof is based [25].

2.2.2 Simple Resource Allocation Problem

If the coefficients on the weight constraint are all set to 1 (i.e., $a_j = 1 \ \ \forall j = 1, 2, \ldots, n$), the problem is reduced to simply selecting b of the items to place into the knapsack. If the objective function allows any concave (on the interval bounded by b), non-linear, increasing function, $f_j(x_j)$ to account for the contribution of each item to the total utility, this problem becomes the Simple Resource Allocation Problem (SRA) [27].

The Simple Resource Allocation Problem is an optimization problem that determines the allocation of a fixed amount of resources to a given number of activities in order to achieve the most effective results and is formulated as follows [24]:

$$\max \sum_{j=1}^{n} f_j(x_j), \tag{2.3}$$

subject to

$$\sum_{j=1}^{n} x_j \leq b$$
$$\mathbf{x} \in \mathbb{Z}^n$$

SRA has been used for both linear and non-linear applications including object detection [4], marketing efforts [32], and portfolio selection [12]. SRA's corresponding decision problem is NP-COMPLETE, when $f_j(x_j)$ is non-linear for all $j = 1, 2, \ldots, n$ [14] [24]. In Chapter 3, a modification of the SRA is used to select a "portfolio" of sensors to assign to various missions.

2.3 Redundant Sensor Assignments

A review of reliability engineering concepts is also necessary to develop the problem formulation outlined in this research.

2.3.1 Structure Functions

Consider the state of an n-component system. First, define an indicator variable E_i, which is equal to 1 if the i-th component of the system is functioning properly and 0 if the i-th component has failed. Also, define an overall system indicator variable, E, for the entire system that is equal to 1 if the entire system is functioning properly and 0 if the system has failed. After determining the disposition of each E_i, the disposition of the entire system can be determined. Therefore, E is a function of E_i for all $i = 1, 2, \ldots, n$; specifically, it is called a *structure function*, and is designated by Φ, i.e., $E = \Phi(E_1, E_2, \ldots, E_n)$ [37].

2.3.2 Series, Parallel, and Mixed Systems

If $\Phi(\boldsymbol{E}) = \prod_{i=1}^{n} E_i$ (i.e., if one component fails, the entire system fails), the system is said to be a *series* system. If $\Phi(\boldsymbol{E}) = 1 - (\prod_{i=1}^{n}(1 - E_i))$ (i.e. the system only fails when all components fail), the system is said to be *parallel redundant*. The combination of both series and parallel redundant components forms a *mixed* system. Such a system that has a definite path is called a *bridge structure network* [37]. A simple *bridge structure network* is shown in Figure 2.2. In this bridge structure, the double index ij is used where i is the index of the parallel components within a given serial stage, and j is the index for the serial stage.

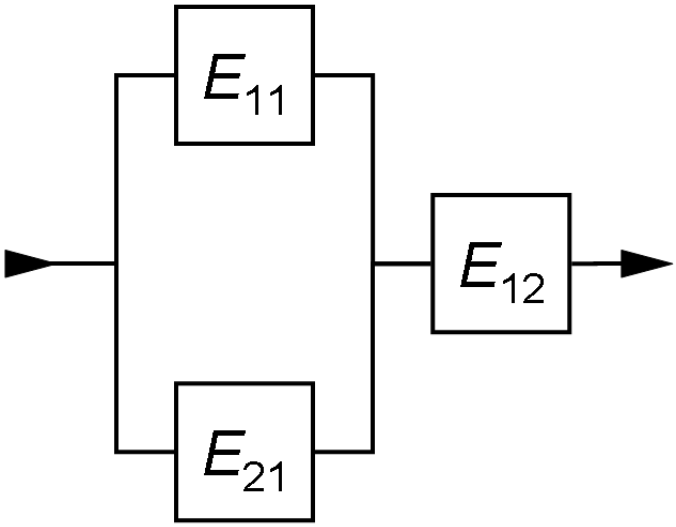

Figure 2.2 A Simple Bridge Structure Network

2.3.3 Probability Calculations for Mixed Systems

Assuming statistical independence of the probability ($P(E_{ij})$) of the individual components succeeding (i.e., the components function independently from one another), the reliability of the system in Figure 2.2 is calculated as follows:

$$R = [1 - (1 - P(E_{11}))(1 - P(E_{21}))][P(E_{12})] \tag{2.4}$$

If a system consists of a process that must follow a sequence of events probabilistically (where components may be assigned in both serial and parallel redundant configurations), it can be modelled and calculated as such [3]. This is shown in Figure 2.3

Mathematically, the reliability of the bridge structure network in Figure 2.3 generalizes into the following form:

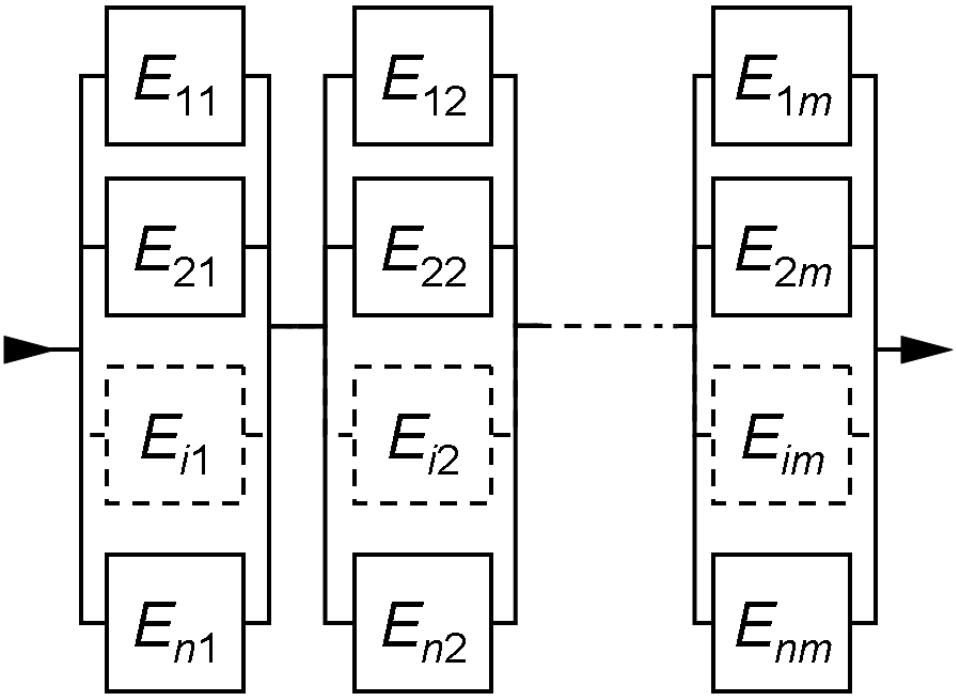

Figure 2.3 A General Bridge Structure Network

$$R = \prod_{j=1}^{m} \left(1 - \prod_{i=1}^{n} (1 - P(E_{ij})) \right) \tag{2.5}$$

For this generalization, i is the index of parallel redundant components and j is the index of stages in serial. The collection of elements in parallel form what is called a *subsystem*, with the probability of success for an individual *subsystem* calculated by the $1 - \prod_{i=1}^{n}(1 - P(E_{ij}))$ portion of Equation 2.5. This method of calculating an overall system probability will be utilized in this research to examine the probabilities of success for various sensor assignments.

2.3.4 Series-Parallel Redundant Allocation Problem

The series-parallel redundant allocation problem describes the problem of maximizing a given system's reliability in a constrained environment [15]. One such problem is stated as follows:

$$\max R(x|\rho) = \prod_{j=1}^{m} \left(1 - \prod_{i=1}^{n} (1 - \rho_{ij})^{x_{ij}} \right) \tag{2.6}$$

subject to

$$\sum_{j=1}^{m} \sum_{i=1}^{n} w_{ij} x_{ij} \leq W$$

$$\sum_{j=1}^{m} \sum_{i=1}^{n} c_{ij} x_{ij} \leq C$$

$$x \in \mathbb{Z}^{m \times n}$$

The objective function in Problem 2.6 is noticeably similar to Equation 2.5. The decision variable, x_{ij} indicates the number of component i assigned to subsystem j. Component i's probability at properly functioning when assigned to subsystem j is denoted by ρ_{ij}. The number of available component choices is denoted by n, and the number of subsystems is denoted by m. The value w_{ij} denotes the weight of assigning component i to subsystem j; the max total weight is W. The value c_{ij} denotes the cost of assigning component i to subsystem j; the max total cost is C.

Solution attempts on a set of 33 problem instances have been made in previous research efforts using surrogate constraints [33], linear approximations [23], and Genetic Algorithms (GA) [5], [6], [41], with the best solutions to date found using a GA methodology [41].

The general form of the series-parallel redundant allocation problem with χ constraints is stated as:

$$\max R(x|\rho) = \prod_{j=1}^{m}\left(1 - \prod_{i=1}^{n}(1-\rho_{ij})^{x_{ij}}\right) \tag{2.7}$$

subject to

$$\sum_{j=1}^{m}\sum_{i=1}^{n} a_{ijk}x_{ij} \leq b_k \;\; \forall\, k = 1 \ldots \chi$$

$$x \in \mathbb{Z}^{m\times n}$$

2.3.5 Multi-Objective Series-Parallel Redundant Allocation Problem

A multi-objective version of the general series-parallel redundant allocation problem exists. In this formulation, reliability is maximized, while cost and weight are minimized, subject to additional structural constraints. The multi-objective series-parallel redundant allocation problem is stated as [36]:

$$\begin{aligned}
\max \quad & R(x|\rho) = \prod_{j=1}^{m}\left(1 - \prod_{i=1}^{n}(1-\rho_{ij})^{x_{ij}}\right) \\
\min \quad & W(x) = \sum_{j=1}^{m}\sum_{i=1}^{n} w_{ij}x_{ij} \\
\min \quad & C(x) = \sum_{j=1}^{m}\sum_{i=1}^{n} c_{ij}x_{ij}
\end{aligned} \tag{2.8}$$

subject to

$$\sum_{j=1}^{m}\sum_{i=1}^{n} a_{ijk}x_{ij} \leq b_k \;\; \forall\, k = 1 \ldots \chi$$

$$x \in \mathbb{Z}^{m\times n}$$

Several methodologies have been applied to this problem, including fuzzy optimization [9], [18], goal programming [17], [18], surrogate constraints [21], dynamic programming [31], and the ε-constraint method [30].

2.4 Comparing Multicriteria Optimization Problems (MOP)

2.4.1 Single Objective Preference and Optimality

In single objective optimization problems, the comparison of one feasible solution to another is accomplished by the scalar comparison operation. In the case of the maximization objective, a solution $\hat{x}$ within the feasible region Ω, is *preferred* to a solution x, if $f(\hat{x}) > f(x)$. Additionally, $\hat{x}$ is said to be *optimal* if there is no other $x \in \Omega$ such that $f(x) > f(\hat{x})$ [34].

2.4.2 Multicriteria Efficiency

In the case of an MOP, the concept of *efficiency* replaces the idea of optimality. Again, in the case of the maximization objective, a feasible solution $\hat{\mathbf{x}} \in \Omega$ is called *efficient* if there is no other $\mathbf{x} \in \Omega$ such that $F(\mathbf{x}) \geq F(\hat{\mathbf{x}})$. The major difference between a single objective problem and an MOP is that $F(\mathbf{x})$ is now a vector-valued function, such that $F(\mathbf{x}) \in \mathbb{R}^p$, and the $\geq$ operator now refers to the component-wise comparisons of the individual vector components of $F(\mathbf{x})$. For example in $\mathbb{R}^4, (3,4,2,1) \geq (2,3,2,1)$ since every element in the left hand vector is greater than or equal to every element in the right hand vector. It should be noted that the component-wise operator does not order the entire space of $\mathbb{R}^p$. Situations could arise in which some elements of a vector are greater than or equal to their corresponding elements in another vector, but this does not hold for all elements. For example, (3,4,2,1) and (4,3,2,1) cannot be ordered using the component-wise operator, since neither vector dominates the other in all four elements [11].

2.4.3 Weighted Sum Scalarization

A common method for ordering feasible MOP solutions is to map from $\mathbb{R}^p \rightarrow \mathbb{R}^1$ via a weighted sum scalarization method [11]. Using this technique, the problem 1.1 is converted to:

$$\max_{\mathbf{x}\in\Omega} \sum_{k=1}^{p} \lambda_k f_k(\mathbf{x}), \tag{2.9}$$
$$\mathbf{x} \in \mathbb{B}^n$$
$$f(\mathbf{x}) \in \mathbb{R}$$

In this formulation, $f_k(\mathbf{x})$ represents the k-th individual component of the p-dimensional multiple objective function, $F(\mathbf{x})$. A mapping of this type allows individual priorities for the various objectives within $F(\mathbf{x})$ to be altered to suit the decision maker's priorities. This will be the main method to adapt the MOP objective function for the sensor assignment problem because of its ease of quickly creating mappings to $\mathbb{R}^1$, and the ability to customize the priorities with the weighted sum coefficients.

2.4.4 Other Methods of Ordering Multi-Objective Space

In addition to weighted sum methods, there are several different ways to order multi-objective space. The ε-constraint method chooses a single objective to be optimized, and the other objectives are transformed into constraints [20]. This method works well if acceptable thresholds for all objectives are well known, however; if the goal is to appropriately share resources among several objectives, the method is not necessarily appropriate.

Compromise points involve minimizing the normalized distance between all points in the efficient set and the ideal point, composed of all individual objectives' optimal solutions [11]. The problem with this method is it requires knowledge of the entire efficient set for comparison purposes, and this is not always practically possible.

2.5 Related Heuristics Methods

This section presents a description of the major heuristics methodologies used later in the research. Both methods, simple ascent and simulated annealing are classified as *threshold algorithms* because they select a neighbor of the current solution (also called the incumbent solution) and compare the incumbent to the neighbor, accepting or rejecting that neighbor according to some type of threshold [2].

2.5.1 Simple Ascent and Neighborhoods

For this research, simple ascent will refer to a simple local search procedure of comparing a given feasible solution to another feasible neighboring solution. The new solution is accepted as the incumbent solution if it has a better objective function value than the current incumbent solution; it is rejected otherwise. In this case, the threshold for acceptance is 0 (i.e., the difference between the new solution and the incumbent solution must be greater than 0). Simple ascent is generally referred to as *iterative improvement* [2]. Pseudocode for this process is presented in Figure 2.4.

INITIALIZATION: Generate a starting solution $\mathbf{x}$.
  Set the initial incumbent solution $\mathbf{x}^* = \mathbf{x}$.
WHILE $i <$ *max iterations* DO
  Choose a *random neighbor* $\mathbf{x}'$ of the current solution.
  If $f(\mathbf{x}') > f(\mathbf{x}^*)$, set $\mathbf{x}^* = \mathbf{x}'$
  $i = i + 1$
  END WHILE
OUTPUT: $\mathbf{x}^*$.

Figure 2.4 Simple Descent Algorithm

If the value of *max iterations* is sufficiently high, the algorithm will eventually converge to a point where no other neighbors have a better objective function value. At this time, it can be said that convergence to a local optimal has happened if the concavity of the region is not guaranteed [34]. The selection of the neighbor also needs to be specified via a *neighborhood function* [2].

2.5.2 Simulated Annealing

Simulated annealing is an algorithm that is analogous to the annealing process of metal [26]. It is a local search algorithm that searches a neighborhood and makes improving moves when the neighbor has a better objective function value than the current incumbent solution. It also allows dis-improving moves in an effort to prevent the trap of a local optima. Such dis-improving moves are probabilistically accepted according to a threshold following an exponential distribution. The distribution is specified by Δ, which is the distance between the incumbent solution and the potential new incumbent, and c, which represents the *temperature* of the process [2].

The *temperature* is initialized at some level, and then decreased by a cooling schedule, which typically follows a geometric progression. This process has both an inner loop and an outer loop. In the outer loop, a *frozen* state of the system is specified by some condition, which can either be a number of iterations or a specific *temperature* level. In the inner loop, an *equilibrium* level is usually specified, which is also often a predetermined number of iterations [2]. General pseudocode for simulated annealing is presented in Figure 2.5.

In simulated annealing, several parameters require value specification. Those parameters include: the initial starting solution $\mathbf{x}$, the starting temperature c, the *frozen* state, the *equilibrium* state, the *random neighbor* selection criteria, and the cooling schedule for c. The goal of these parameter selections is to find the best method to allow dis-improving moves early enough in the algorithmic process to avoid stagnation in a local optima, and then transition to allowing only improving moves later in the algorithmic process. The drawback to simulated annealing is that the determination of appropriate values for these parameters often requires a large amount of experimentation, especially when applied to the series-parallel redundant allocation problem [28].

INITIALIZATION: Generate a starting solution $\mathbf{x}$.
  Set the initial incumbent solution $\mathbf{x}^* = \mathbf{x}$.
  Determine a starting temperature c.

WHILE not yet *frozen* DO
  WHILE not yet at *equilibrium* for this temperature DO
    Choose a *random neighbor* $\mathbf{x}'$ of the current solution.
    Set $\Delta = f(\mathbf{x}') - f(\mathbf{x})$
    If $\Delta \geq 0$ (uphill move)
      Set $\mathbf{x} = \mathbf{x}'$
      If $f(\mathbf{x}) > f(\mathbf{x}^*)$, set $\mathbf{x}^* = \mathbf{x}$
    Else (downhill move)
      Choose a random number r uniformly [0,1].
      If $r < e^{-\Delta/c}$, set $\mathbf{x} = \mathbf{x}'$
    END 'WHILE not at *equilibrium*' loop.
  Lower the temperature c
  END 'WHILE not yet *frozen*' loop.
OUTPUT: $\mathbf{x}^*$.

Figure 2.5 Simulated Annealing Algorithm

2.5.3 Other Heuristics Techniques

Genetic algorithms is a heuristics technique that involves creating a random population of solutions and then combining those solutions via a crossover operator that combines the properties of two or more parent solutions. Mutations can also be introduced to ensure diversity [22]. While genetic algorithms have produced the best solutions for the single objective series-parallel reliability problem [41], this research seeks to produce quality multi-objective solutions quickly on desktop computers, and simulated annealing was selected since it generally requires less objective function evaluations. Large populations can require, in a worst case scenario, a large amount of objective function evaluations, making the algorithm very computationally demanding.

The tabu search procedure involves creating a tabu list of previously visited solutions to avoid revisiting them and allows for some more efficient searching of the feasible region [19]. While tabu search can also produce high quality solutions, an

easy to implement method was desired for this research, but that does not preclude exploring the use of Tabu search in future research.

2.6 Conclusion

This chapter reviewed the relevant literature to the techniques which are developed and presented in Chapter 3. It is by employing various aspects of these concepts that a cohesive model to solve the sensor assignment problem is created.

3. Methodology

This chapter begins by describing the construction of the Maximum Utility Sensor Assignment Problem (MUSAP) under the independence of the sensor chain assumptions which were stated in Chapter 1. Following that, the decision version of MUSAP is hypothesized to be NP-COMPLETE. This hypothesis is strengthened by showing it to be an instance of the Simple Resource Allocation (SRA) problem. Due to the intractability of the MUSAP problem in larger instances, the various heuristic techniques that were used to solve MUSAP as well as the methodologies used to test them are described.

3.1 Formulation of the Maximum Utility Sensor Assignment Problem (MUSAP)

3.1.1 Decision Variables and Probability Settings

The key decision for this research is which sensors to assign to particular missions during various stages in the decision process. This will be signified by $x \in \mathbb{B}^{\alpha \times \beta \times \chi}$ where α is the total number of available sensors, β, is the total number of missions, and χ is the number of stages in the decision process. Therefore,

$$x_{ijk} \equiv \begin{cases} 1 & \text{if sensor } i \text{ is assigned to mission } j, \text{ at stage } k \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

The probabilities of success for a particular sensor i to perform mission j at stage k, given the sensor was assigned, is designated by ρ_{ijk}. Since the probability of success is only greater than zero if the sensor is actually assigned, this probability can be represented by the expression $\rho_{ijk}x_{ijk}$. The bridge structure network for a specific mission j is shown in Figure 3.1.

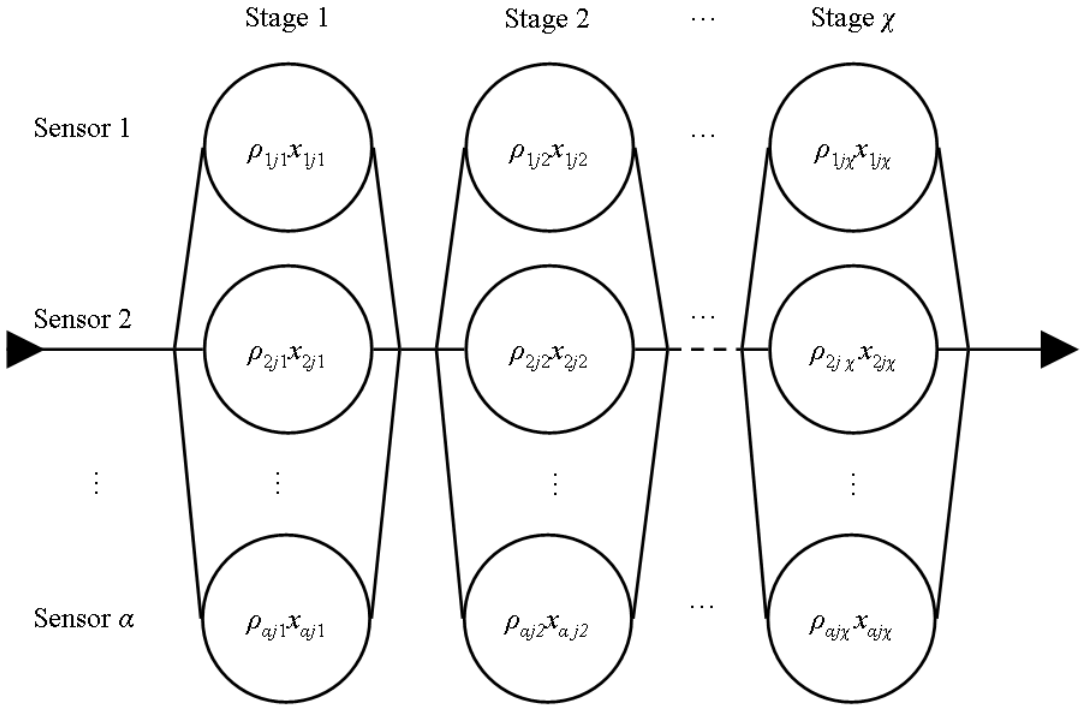

Figure 3.1 General Bridge Structure Network for MUSAP (for a mission j)

3.1.2 Objective Function Construction

It is possible to construct an overall probability of success for a given mission j at stage k, under the assumption that several sensors can be assigned in parallel to that mission j at stage k. This probability is derived from the reliability theory discussed in Chapter 2 and is given in Equation 3.2:

$$h_k(\mathbf{x}) = 1 - \prod_{i=1}^{\alpha}(1 - \rho_{ijk}x_{ijk}) \;\; \forall\, j = 1, 2, \ldots, \beta. \tag{3.2}$$

This probability is the compliment of the probability of every single assigned sensor failing. Put another way, it is the probability that at least one assigned sensor accomplishes that a given mission's stage.

The individual stages are independent, serial tasks; thus, the overall probability of success for mission j across all stages is simply the product of the probability of success of the individual stages, shown in Equation 3.3:

$$f_j(\mathbf{x}) = \prod_{k=1}^{\chi} \left(1 - \prod_{i=1}^{\alpha} (1 - \rho_{ijk} x_{ijk}) \right) = \prod_{k=1}^{\chi} h_k(\mathbf{x}). \tag{3.3}$$

Under the assumption that each of these missions has a weight, or significance, w_j, the overall objective function can now be stated in Expression 3.4:

$$\sum_{j=1}^{\beta} w_j \left(\prod_{k=1}^{\chi} \left[1 - \prod_{i=1}^{\alpha} (1 - \rho_{ijk} x_{ijk}) \right] \right) = \sum_{j=1}^{\beta} w_j f_j(\mathbf{x}). \tag{3.4}$$

3.1.3 Constraint Formulation

To ensure feasible sensor assignments, some structural constraints are necessary. The first set of constraints ensures that it is not possible to assign more sensors than are available at a given stage. Specifically,

$$\sum_{i=1}^{\alpha} \sum_{j=1}^{\beta} x_{ijk} \leq \alpha \ \ \forall \, k = 1, 2, \ldots, \chi. \tag{3.5}$$

The second set of constraints prevents tasking sensors to more than one mission at a given stage. Therefore,

$$\sum_{j=1}^{\beta} x_{ijk} \leq 1 \ \ \forall \, i = 1, 2, \ldots, \alpha, \ \ k = 1, 2, \ldots, \chi. \tag{3.6}$$

3.1.4 Complete Mathematical Formulation of MUSAP

Combining the objective function and constrainst into a concise, mathematical formulation produces:

$$\max \sum_{j=1}^{\beta} w_j \left(\prod_{k=1}^{\chi} \left[1 - \prod_{i=1}^{\alpha} (1 - \rho_{ijk} x_{ijk}) \right] \right) \tag{3.7}$$

subject to:

$$\sum_{i=1}^{\alpha} \sum_{j=1}^{\beta} x_{ijk} \leq \alpha \quad \forall \quad k = 1, 2, \ldots, \chi$$

$$\sum_{j=1}^{\beta} x_{ijk} \leq 1 \quad \forall \quad i = 1, 2, \ldots, \alpha, \quad k = 1, 2, \ldots, \chi$$

$$x_{ijk} \in \{0, 1\} \quad \forall \quad i = 1, 2, \ldots, \alpha, \quad j = 1, 2, \ldots, \beta, \quad k = 1, 2, \ldots, \chi.$$

3.2 NP-Completeness of Maximum Utility Sensor Assignment Problem

It is hypothesized that MUSAP is NP-COMPLETE, as it can be simplified to a Simple Resource Allocation Problem.

3.2.1 MUSAP as a Simple Resource Allocation Problem

The decision problem associated with MUSAP is as follows:

Given a set of weights, w_j; a multidimensional set of probabilities, ρ_{ijk}; a max number of sensors, α; a max number of missions, β; and a max number of stages, χ; and a real-valued objective function target, F; does there exist a binary assignment

to $\mathbf{x} \in \mathbb{B}^{\alpha \times \beta \times \chi}$ such that

$$\sum_{j=1}^{\beta} w_j \left(\prod_{k=1}^{\chi} \left[1 - \prod_{i=1}^{\alpha} (1 - \rho_{ijk} x_{ijk}) \right] \right) \geq F \tag{3.8}$$

$$\begin{aligned}
\sum_{i=1}^{\alpha} \sum_{j=1}^{\beta} x_{ijk} &\leq \alpha \quad \forall \quad k = 1, 2, \ldots, \chi \\
\sum_{j=1}^{\beta} x_{ijk} &\leq 1 \quad \forall \quad i = 1, 2, \ldots, \alpha, \quad k = 1, 2, \ldots, \chi \\
x_{ijk} &\in \{0, 1\} \quad \forall \quad i = 1, 2, \ldots, \alpha, \quad j = 1, 2, \ldots, \beta, \quad k = 1, 2, \ldots, \chi.
\end{aligned}$$

The decision problem associated with MUSAP will be transformed into an instance of the SRA problem, and SRA is known to be NP-COMPLETE.

Given an instance of SRA (Decision Problem) with b, G, and a set of concave, increasing, non-linear functions, f_p (where there are a total of γ functions), does there exist an integer assignment to $y \in \mathbb{Z}^{\gamma}$ such that

$$\begin{aligned}
\sum_{p=1}^{\gamma} f_p(y_p) &\geq G \\
\sum_{p=1}^{\gamma} y_p &\leq b
\end{aligned} \tag{3.9}$$

First, assign $F = G$ and $\alpha = b$. Assign $\chi = 1$; this collapses the set of constraints into one constraint (the set k will not be mentioned for the rest of this section). $\sum_{i=1}^{\alpha} x_{ij}$ corresponds to y_p (the sum of many binary variables is an integer). $\gamma = \beta$ and $\sum_{p=1}^{\gamma} y_p = \sum_{i=1}^{\alpha} \sum_{j=1}^{\beta} x_{ijk}$, and the expression $(\prod_{k=1}^{\chi} [1 - \prod_{i=1}^{\alpha} (1 - \rho_{ijk} x_{ijk})])$ collapses to $[1 - \prod_{i=1}^{\alpha} (1 - \rho_{ij} x_{ij})]$.

The next step involves showing that $w_j [1 - \prod_{i=1}^{\alpha} (1 - \rho_{ij} x_{ij})]$ is a concave, increasing, non-linear function, $f_p(y_p)$. For a given mission, j, assign all ρ_{ij} to the same

value (ρ_{ij} for different missions can still be different). The specific determination of ρ_{ij}, and w_j is such that the function $f_p(y_p)$ is properly represented.

By setting ρ_{ij} constant to a value, ε_j for a given mission, and setting w_j equal to some scalar, c_j/ε_j, any concave, increasing, non-linear function can be approximated within a desired degree of precision. This is demonstrated in Figure 3.2. In this case, the scalar, c_j is held constant at 1; however it may take on any value required to scale the function, just as the value ρ_{ij} may take on any value to shape the function.

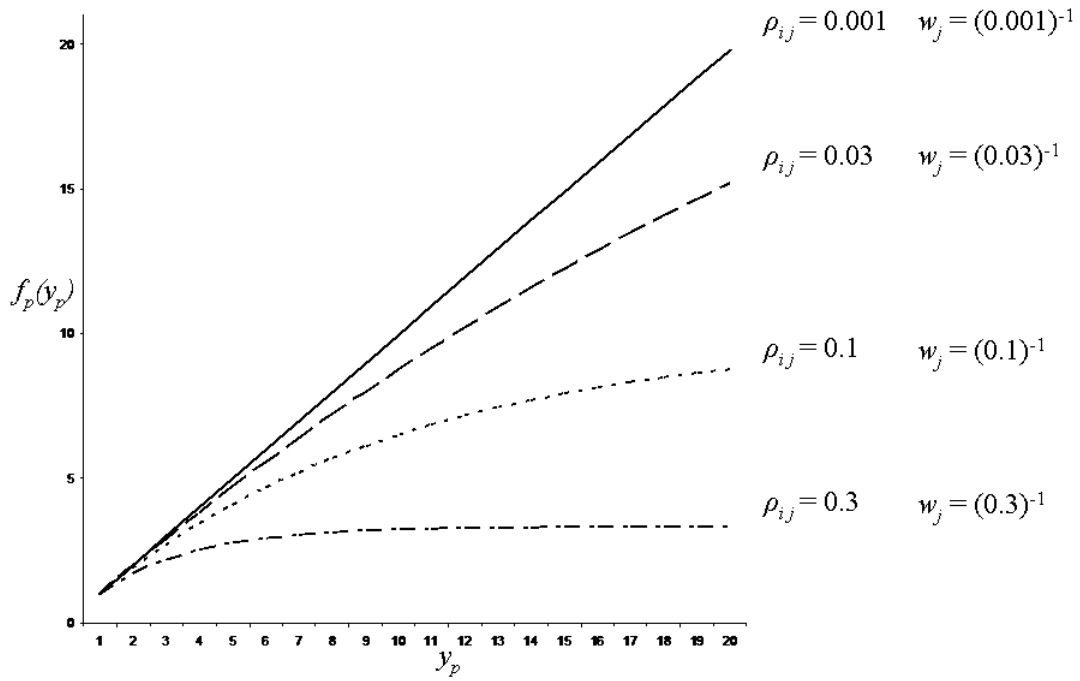

Figure 3.2 Adjusting ρ_{ij} and w_j to create any concave, increasing, non-linear function

As more sensors are tasked against a mission (setting more $x_{ij} = 1$), the concave, increasing, non-linear contribution to the objective function is created, just as allocating more resources in SRA (increasing the value of y_p) accomplished a similar increase in the form of the function $f_p(y_p)$.

It should be noted that this alone does not prove that MUSAP is NP-COMPLETE, but rather, strongly suggests it, as this simplification of MUSAP is a type of Resource Allocation Problem. In order to accomplish that, it would be required to show that

any function $f_p(y_p)$ could be approximated in polynomial time to any desired degree of precision by selection of coefficients ρ_{ij} and w_j.

3.3 Solving MUSAP to Optimality

It is important to solve some problem instances of MUSAP to optimality in order to have a bases for comparison for the heuristics techniques that will be later discussed. Since MUSAP is non-linear, a Linear Programming (LP) relaxation of MUSAP is not available. Thus, Integer Programming solution methodologies (i.e.: Branch and Bound, Branch and Cut, and Cutting Planes) cannot be used. Lagrangian relaxations are also ruled out due to the complexity of computing the partial derivatives for this formulation. Dynamic Programming has a similar problem due to the high dimensionality of the set of decision variables [35]. The technique that remains is explicit enumeration.

3.3.1 Enumeration Methodology

The stages can be solved as individual stage problems, $(g_k(\mathbf{x}) \ \ \forall k = 1, 2, \ldots, \chi)$ and the product of the individual mission probabilities at each stage can be used to determine the total mission probability. This is possible due to the independence of stage probabilities assumption that was previously stated. This drastically reduces the number of combinations which would require examination if this assumption was not used.

At a given stage, with α available sensors, and β missions that they may be assigned, there are β^α feasible configurations. Each configuration may be examined, evaluating its objective function value for each of the $k = 1$ to χ stages. This stage function is designated $g_k(\mathbf{x})$. The independence assumption implies that this type of search is sufficient. However, if it is necessary to search all possible configurations, dependant upon the assignments in all stages, that number becomes $(\beta^\alpha)^\chi$.

The enumeration process used assumes the independence of stages, and explicitly enumerates each of the β^{α} feasible configurations and conducts the stage objective function evaluation for $g_k(\mathbf{x})$, storing the best configurations by stage. The best stage configurations are reassembled into the overall best configuration. Pseudocode for the enumeration process is presented in Figure 3.3.

```
INITIALIZATION: Assign all x_ijk = 0
  Assign the initial best configuration x*_k = 0  ∀k = 1, 2, ..., χ
  Assign the initial best solution f* = 0
  Assign the initial best stage solution g*_k = 0  ∀k = 1, 2, ..., χ
FOR j(1) = 1 to β
  FOR j(2) = 1 to β
    FOR j(3) = 1 to β
      Continue nesting j(i) until i = α
        FOR i = α
          ASSIGN x_{i,j(i),•} = 1
          FOR k = 1 to χ
            EVALUATE g_k(x_k)
            IF g_k(x_k) > g*_k THEN g*_k = g_k(x_k), x*_k = x_k
          END
        END
      ...
    END
  END
END
OUTPUT: x* (all x*_k aggregated)
OUTPUT: f(x*) (best solution using all best stage configurations).
```

Figure 3.3 Enumeration Algorithm

3.3.2 Intractability of Full Enumeration

Even with the independence assumption, β^{α} still has an exponential relationship to the problem size, making it intractable. The number of function evaluations that are required for the test problem sizes used in this research is shown in Table 3.1 (for simplification purposes, only 4 stages are tested, since there is a direct analogy of 4 stages to the real world problem). Due to the exponential growth in the number

of points that must be considered, only the smallest two test problem sizes are solved using this method.

Table 3.1 Number of Feasible Points for Various Problem Sizes

sensors	missions	stages	number of points
10	4	4	1.05×10^{6}
10	7	4	2.82×10^{8}
20	8	4	1.15×10^{18}
20	14	4	8.37×10^{22}
30	12	4	2.37×10^{32}
30	21	4	4.64×10^{39}
40	16	4	1.46×10^{48}
40	28	4	7.70×10^{57}

3.4 Approximating MUSAP Using Heuristic Methods

3.4.1 Initial Starting Points

The heuristic methods utilized require feasible starting points in order to begin. Two construction techniques for creating those starting points were examined. Both techniques use a greedy approach, but differ in what is rewarded "greedily". The idea of beginning with a greedy or semi-greedy starting point and then applying a local search technique is called the Greedy Randomized Adapted Search Procedure (GRASP) [13].

The first technique, called *greedy individual*, follows the basic idea that "everyone works where they do best." With this technique, each sensor is considered at each stage in the decision process. A sensor i is assigned to a mission j during stage k if its probability of detection, ρ_{ijk} is greater than its probability of detection for any other mission j, at that stage. The advantage of this technique is that it constructs a starting point in few mathematical operations. The pseudocode for *greedy individual* is shown in Figure 3.4.

```
INITIALIZATION: Assign all x_ijk = 0
FOR k = 1 to χ
  FOR j = 1 to β
    FOR i = 1 to α
      IF ρ_ijk =MAX(ρ_i•j) THEN x_ijk = 1
    END
  END
END
OUTPUT: Feasible assignment of x
```

Figure 3.4 Greedy Individual Construction Algorithm

The second technique, called *greedy marginal*, follows the basic idea that "everyone works where their marginal contribution is highest." With this technique (as with *greedy individual*), each sensor is considered at each stage in the decision process. The difference between the two techniques is that a sensor i is assigned to mission j during stage k if its contribution to the objective function is greatest. This technique takes longer to construct, as it requires a partial objective function evaluation for each potential assignment during the construction phase. The pseudocode for *greedy marginal* is shown in Figure 3.5.

```
INITIALIZATION: Assign all x_ijk = 0
  Assign L_k = the set of unassigned sensors i for a given stage k
  Assign the initial best configuration x*_k = 0  ∀k = 1, 2, ..., χ
  Assign the initial best stage solution g*_k = 0  ∀k = 1, 2, ..., χ
FOR k = 1 to χ
  FOR j = 1 to β
    Randomly Choose i ∈ L_k
    x_ijk = 1 , L_k = L_k/i
  END
  EVALUATE g_k(x) , ASSIGN g*_k = g_k(x)
END
FOR k = 1 to χ
  FOR j = 1 to β
    FOR i = 1 to α
      IF i is unassigned, THEN EVALUATE g_k(x) for j = 1 to β,
      ASSIGN x_ijk = 1 such that g_k(x) is the best of evaluated solutions, L_k = L_k/i
    END
  END
END
OUTPUT: Feasible assignment of x
```

Figure 3.5 Greedy Marginal Construction Algorithm

3.4.2 Neighborhood Functions

A simple exchange network is used to generate feasible neighbors during the iteration process. Essentially, a stage and sensor are picked at random, and then the sensor is swapped to another randomly chosen mission within that stage. The number of swaps that are performed is determined by the size of the neighborhood. In this research, 1-, 2-, and 3-swap neighborhoods are considered as starting neighborhoods (used at the beginning of the iteration scheme) with 1-, 2-, 3-, and 4-swap neighborhoods being considered as ending neighborhoods (used at the end of the iteration scheme). The psuedocode for an n-swap neighborhood function is shown in Figure 3.6. In this scheme, a sensor, i, and a stage, k, are randomly chosen, its current assignment is deleted, and it is randomly assigned to a new mission, j. It is possible (but improbable) that the output will be the exact same configuration as the input configuration. In order to guarantee a true swap, a comparison would

be required to ensure the new sensor assignment was not the same previous assignment. Since this would involve more mathematical operations in order to prevent an improbable event, the check was not included.

INPUT: A current feasible configuration, $\mathbf{x}$, number of swaps, n
FOR $m = 1$ to n
  Randomly choose k (stage)
  Randomly choose i (sensor)
  Set $x_{i \bullet k} = 0$
  Randomly choose j (mission)
  Set $x_{ijk} = 1$
END
OUTPUT: Feasible assignment of $\mathbf{x}$

Figure 3.6 n-swap Neighborhood Function

The degree of *intensification* and *diversification* is also one of the major experimental factors examined by this research. Intensification implies searching in neighborhoods that are very close to the incumbent point, whereas diversification implies searching in neighborhoods that are farther from the incumbent point [19]. These concepts are relative to other neighborhoods. For example, a 2-swap neighborhood is more intense than a 3-swap neighborhood, but it is more diverse than a 1-swap neighborhood.

In Figure 3.7 (assuming a maximization objective), at the point $\mathbf{x}^1$, using the more intense neighborhood, $n(1)$, no improving point can be found. If the more diverse neighborhood, $n(2)$ is utilized, the improving point $\mathbf{x}^2$ can be found. However, once the point $\mathbf{x}^2$ is reached, the neighborhood $n(1)$ can once again find improving moves. Since the perimeter of $n(2)$ is bigger than the perimeter of $n(1)$, it is easier to find improving moves (when they exist), because there are less points to search. Once the algorithm has converged to a local optimal, diverse neighborhoods are required to escape.

The size of the neighborhood determines the degree of intensification and diversification that is going on in the heuristic at that point of time. For the purposes

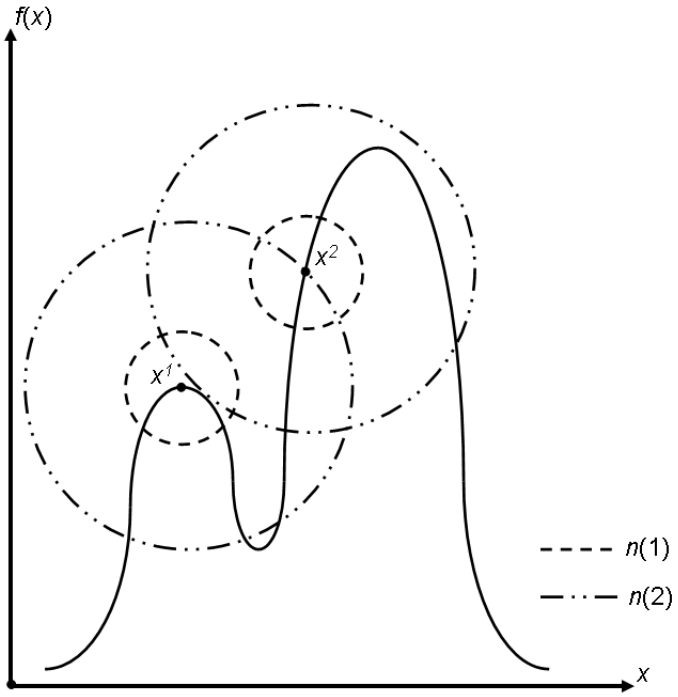

Figure 3.7 Simple Example of Various Neighborhood Sizes

of this research 1-swap neighborhoods are the most intense neighborhoods that will be used; 4-swap neighborhoods are the most diverse neighborhoods. The right mix between intensification and diversification is crucial in order to converge to high quality, near optimal solutions. Three types of diversification strategies are examined: switching to the next bigger neighborhood after 50%, or 75%, or never diversifying. This implies that the diversification parameter d is set to 0.5, the first 50% of the iterations are performed using an n-swap neighborhood, and the remaining iterations are performed using an $(n+1)$-swap neighborhood.

3.4.3 Simple Ascent Applied to MUSAP

The simple ascent algorithm described in Figure 2.4 was tested on all the problem sizes shown in Figure 3.1. Simple ascent also served as a control group to compare against simulated annealing algorithms that were limited to the same number of iterations. In order to ensure that the algorithm's performance was not due

to the uniqueness of a specific problem, ten problem instances for each of the eight problem sizes were constructed (for a total of eighty problems). One hundred replications were performed to give the results greater statistical merit. The pseudocode for simple ascent applied to MUSAP is shown in Figure 3.8.

INITIALIZATION: Generate a starting solution $\mathbf{x}$
INPUT neighborhood size n, diversification rule d, *max iterations*
ASSIGN the initial incumbent solution $f^* = f(\mathbf{x})$
WHILE *count* $<$ *max iterations* DO
  IF *count* $< d \cdot$(*max iterations*) THEN $s = n$, ELSE $s = n + 1$
  Choose a *random neighbor* $\mathbf{x}'$ of the current solution using a neighborhood of size s
  IF $f(\mathbf{x}') > f(\mathbf{x}^*)$, set $\mathbf{x}^* = \mathbf{x}'$
  count = *count* $+ 1$
  END WHILE
OUTPUT: $\mathbf{x}^*$, $f(\mathbf{x}^*)$.

Figure 3.8 Simple Descent Algorithm Applied to MUSAP

3.4.4 Simulated Annealing Applied to MUSAP

The simulated annealing algorithm described in Figure 2.5 was also tested on all problem sizes shown in Table 3.1. Initial temperature was calculated as the distance from the starting point and the *ideal* point. The *ideal* point is defined as the point at which all individual objectives are maximized [11]. In this case, it is the point in which each mission has the highest probability of success, achieved by assigning all sensors to that mission alone. This will be as close to 1 as is possible with asymptotic convergence. For this research, a value of 1 is used for each mission value's ideal state which assumes equal weights of $w_j = 1$, so the ideal objective function value for the weighted sum function is β, which implies that all missions succeeded with probability 1.

The pseudocode for simulated annealing applied to MUSAP is shown in Figure 3.9. The inner *equilibrium* was determined by an inner loop number of iterations, and the *frozen* state was determined by an outer loop number of iterations, that exceeded the inner loop iterations by a factor of 10. Many different techniques

INITIALIZATION: Generate a starting solution $\mathbf{x}$
INPUT neighborhood size n, diversification rule d, cooling parameter γ, *inner iterations*
ASSIGN the initial incumbent solution $f^* = f(\mathbf{x})$, initial temperature $c = \beta - f^*$
ASSIGN (*outer iterations*) $= 10 \cdot$ (*inner iterations*)
ASSIGN *max iterations* = (*inner iterations*) $\cdot$ (*outer iterations*)
FOR *outer* $= 1$ to *outer iterations*
  FOR *inner* $= 1$ to *inner iterations*
    IF *count* $< d \cdot$(*max iterations*) THEN $s = n$, ELSE $s = n + 1$
  Choose a *random neighbor* $\mathbf{x}'$ of the current solution using a neighborhood of size s
    $\Delta = f(\mathbf{x}') - f(\mathbf{x})$
    IF $\Delta \geq 0$ (uphill move)
      Set $\mathbf{x} = \mathbf{x}'$
      If $f(\mathbf{x}) > f(\mathbf{x}^*)$, set $\mathbf{x}^* = \mathbf{x}$
    ELSE (downhill move)
      Choose a random number r uniformly [0,1].
      If $r < e^{-\Delta/c}$, set $\mathbf{x} = \mathbf{x}'$
    END inner loop.
  $c = \gamma \cdot c$
  END outer loop.
OUTPUT: $\mathbf{x}^*$, $f(\mathbf{x}^*)$.

Figure 3.9 Simulated Annealing Applied to MUSAP

for setting inner and outer loop iterations have been attempted [1], [7], [10], [38]. Unfortunately, simulated annealing has performance ambivalence with respect to application, implying a set of parameters the works well for one type of problem has no guarantee of working for other problems [2]. It is for this reason that the factor of 10 of outer loop iterations exceeding inner loop iterations was assigned, as it allows more cooling iterations for larger sized problems on a proportional basis. The cooling schedule was exponential, implying that at every outer loop iteration, the temperature was multiplied by a cooling parameter, γ [2]. The value of γ is an experimental factors examined by this research.

3.4.5 Experimental Design

The experimental factors tested in this research are listed in Table 3.2. The total number of potential configurations is 72 (3 neighborhood types $\times$ 4 settings for

$\gamma \times 2$ starting points constructions $\times$ 3 diversification rules). These 72 combinations are explicitly stated in Appendix A.

Table 3.2 Experimental Factor Settings

Experimental Factors			
neighborhood	γ	**starting point**	**diversify after**
1-swap	0 (Simple Ascent)	marginal	50%
2-swap	0.9 (Simulated Annealing)	individual	75%
3-swap	0.8 (Simulated Annealing)	-	never
-	0.7 (Simulated Annealing)	-	-

These factor levels are very general in order to establish a baseline and were chosen after a smaller screening experiment determined that they may be the bounds of interest for the parameter settings.

3.5 Conclusion

This chapter outlined all of the analytical techniques that were used in this research. Their performance and implementation is detailed in Chapter 4.

4. Implementation and Results

This chapter outlines the implementation of the algorithms described in Chapter 2, specific results by problem type, and overall results trends.

4.1 Preliminary Algorithm Settings

In order to evaluate the 72 algorithms from Tables A.1, A.2, and A.3, ten random problem instances were constructed for each of the the eight problem sizes shown on Table 3.1, for a total of 80 problem instances. To generate a problem instance, a random number between 0 and 1 was assigned to each of the values for ρ_{ijk}. This assumes that each sensor has at least some probability of being able to perform every stage of every mission. Since the algorithms have probabilistic iteration schemes, 100 attempts are made with each of the 10 problem instances for each of the 8 problem sizes, for a total of 1000 attempts to solve each problem size per algorithm.

4.1.1 Evaluation Criteria

The main evaluation criteria is the percentage of the 1000 attempts in which the algorithm provides solutions within 1%, 5%, and 10% of the optimal solution (if known) or the overall best solution found for that problem instance. These are tested against the various $\alpha \times \beta \times \chi$ sized problems designated on Table 3.1, where α is the number of sensors, β is the number of missions, and χ is the number of stages in the problem as specified in Chapter 3. For example a problem size designated 10×4×4 has 10 sensors, 4 missions, and 4 stages.

4.1.2 Sensor Saturated vs. Sensor Sparse Networks

A distinction is made between the sensor saturated and sensor sparse networks. The problem sizes 10×4×4, 20×8×4, 30×12×4, and 40×16×4 are designated as sensor saturated, since there are 2.5 sensors available for every mission. Whereas, the problem sizes 10×7×4, 20×14×4, 30×21×4, and 40×28×4 have approximately 1.43 sensors available for every mission. This density difference results in significant variation in computational requirements. For example, assuming two sensors of equal probability for mission stage success of 0.5, assigning one sensor to mission A has a probability of success of 0.5, but assigning two sensors to that mission would give is a probability of success of 0.75.

4.1.3 Solvable Problems

Through explicit enumeration, the smallest problem sizes from Table 3.1 were solved (the 10×4×4 and 10×7×4 problem instances) with all values of $w_j = 1$, which implies that each mission is of equal importance. Thus, the optimal solution will be the solution that equally distributes the sensors amongst the missions. As indicated in Chapter 3, this required 1.05×10^6 and 2.82×10^8 objective function evaluations. Due to intractability, larger problem sizes were not attempted. The optimal solutions for these 20 problems are used as basis for comparison to assess the performance of the algorithms. For the 60 larger problem sizes, the overall best answer found is used in place of the optimal solution for a comparison basis. The optimal solutions are shown in Table B.1.

4.1.4 Initial Construction Settings

Analysis of the smallest problem sizes, (10×4×4 and 10×7×4) demonstrates that the more sensor-sparse problem of 10×7×4 has a greater difficulty in finding quality solutions on a regular basis, as shown in Tables B.2, B.3, B.4, and B.5.

It is hypothesized that the more sparse sensor networks have a greater tendency to get stuck in local optimal because many local optima are located when a mission is set to automatically fail. This becomes more evident when sparse sensor networks are being optimized; however, it is a possibility given any scenario.

It is possible for a solution to find a neighbor that may improve the overall objection function value, but do so by failing to accomplish one of the stages for a mission, thereby forcing the mission to automatically fail. This will result in the objective function contribution for all sensors assigned to the failed mission to be zero. In subsequent iterations of the algorithm, all the neighbors of that solution which remove any sensors from the failed mission stages will always produce better objective function values, by converting a sensor objective function contribution of zero to a positive value. Once the algorithm has assigned a particular sensor configuration that allows for a failure of a mission, it has a high probability for continuing to remove sensors away from that mission. This phenomenon has the effect of creating numerous local optima.

In the case of the greedy individual construction, the initial constructions do not even guarantee that all missions will be attempted. Thus, with these constructions, the algorithm does not produce quality solutions very frequently. In contrast to the greedy individual constructions, the greedy marginal constructions are designed to require all missions to succeed. As a result, it was required to modify the algorithms so that automatic failure of missions are strictly prohibited.

To test the efficacy of these modifications, the $10\times4\times4$ and $10\times7\times4$ problem sizes were re-accomplished, producing the results in Table B.6, Table B.7, Table B.8, and Table B.9 respectively. These results demonstrate that especially in sensor sparse networks, better quality solutions can be produced by these methods. Additionally, a large degree of local optima can be avoided. It also has the effect of eliminating 36 of the 72 algorithms from consideration, as only algorithms with greedy marginal constructions guarantee avoidance of this particular type of local optimal. The

algorithms removed from consideration are the algorithms with the greedy individual starting points (13-24, 37-48, and 61-72).

For the purposes of implementation, it was necessary to track the probability of success for the individual missions with a mission function, $h_j(\mathbf{x})$. The algorithms in Figure 3.8 and Figure 3.9 have not changed, with the exception of any move from the incumbent solution also having the additional requirement of $\min_j(h_j(\mathbf{x})) \neq 0$, which explicitly states that no moves that allow automatic mission failures are allowed.

4.1.5 Iteration counts for Inner Loop Equilibrium

With these new algorithms, the inner loop iteration counts were then specified. For saturated networks, the number of sensors was used for the inner loop iterations, since it was desirable to link the inner loop iterations to the problem size parameter. This is because it is reasonable to assume that larger problems would require more iterations to properly explore the space. The final iteration counts are shown in Table 4.1.

Table 4.1 Number of Inner, Outer and Total Iterations for Various Problem Sizes

size	inner	outer	total
10×4×4	10	100	1000
10×7×4	28	280	7840
20×8×4	20	200	4000
20×14×4	39	390	15210
30×12×4	20	300	9000
30×21×4	43	430	18490
40×16×4	40	400	16000
40×28×4	46	460	21160

For sparse networks, iteration counts were varied (beginning with the number of sensors in the problem) to determine the number of iterations for which the best algorithm continues to produce at least 75% of final solutions within 10% of the overall best solution found (or the true optimal in the case of the 10×4×4 and 10×7×4 sized problems). The algorithms were run with a relatively large number

of inner loop iterations (twice the number of sensors) to establish the best solutions to be used as comparison bases for subsequent problems.

4.2 Results by Problem Size

4.2.1 Problem Size 10×4×4

The results for Problem Size 10×4×4 are in Table B.6. Recall that for this problem, the optimal solution was first found using explicit enumeration; all 36 algorithms produced solutions within within 10% of the optimal, with the worst being Algorithm 27 at 98.8% of the trials within 10% of optimal, and the best being tied between Algorithms 9, 29, 30, 33, 53, 56, and 57, all at 100% of the trials within 10% of the optimal. These excellent solutions result from the fact that the sensor saturated networks have numerous acceptable configurations and less local optima. When the evaluation criteria required solutions to be within 5% of optimal criteria, all 36 algorithms still perform fairly well, with Algorithm 10 the worst at 84.5% and Algorithm 57 the best at 96.8%. Lastly, when requiring solutions to be within 1% of optimal, Algorithm 10 performed the worst at 2.2%, and Algorithm 53 performed the best at 25.9%.

With the aggregated results in Table B.7, several observations with respect to this problem size can be made about the different parameter settings' efficacy. The diversification rule of "never" actually allowing diversification outperforms the other diversification settings (within 10%, 99.7% of the time; within 5%, 92.2% of the time; and within 1%, 17.1% of the time). Diversifying after 50% of the iterations is the worst diversification setting.

The 2-swap neighborhood outperforms the other two neighborhood functions, producing solutions within 10% of the optimal 99.9% of the time; within 5%, 93.5% of the time; and within 1%, 18.2% of the time. The best parameter setting for γ,

is 0 (implying Simple Ascent) which produced solutions within 10% of the optimal 99.7% of the time, within 5%, 94.5% of the time, and within 1%, 17.3% of the time.

4.2.2 Problem Size 10×7×4

The results for problem size 10×7×4 by algorithm are displayed in Table B.8. For the evaluation criteria of producing solutions within 10% and within 5% of the optimal solution, Algorithm 54 performed best (76.3% and 35.0%) and Algorithm 49 performed the worst (33.7% and 10.2%). When the criteria required solutions to be with 1% of the optimal, Algorithm 55 performed best (2.5%), and Algorithm 28 is the worst (0.1%).

The aggregated results for problem size 10×7×4 in Table B.9 show that the best diversification rule is to "never" diversify; this technique produces solutions within 10% of the optimal 61.3% of the time, within 5%, 24.4% of the time, and within 1%, 1.2% of the time. The 2-swap neighborhood is the preferred neighborhood function which, producing solutions within 10%, 71.0% of the time, within 5%, 31.2% of the time, and within 1%, 1.5% of the time. The best parameter setting for γ is 0.9 for 10% and 5% of the optimal, achieving values of 63.9% and 23.4% respectively; however, a γ setting of 0 (Simple Ascent) produces the best results when the criteria requires solutions to be within 1% of optimal, achieving it 1.1% of the time.

4.2.3 Problem Size 20×8×4

The results for problem size 20×8×4 by algorithm are in Table B.10. For the evaluation criteria of producing solutions within 10% and within 5% of the best solution found thus far, all of the algorithms perform well (above 94%). When the criteria required solutions to be with 1% of the best solution found thus far, Algorithm 49 is the best (19.8%), and Algorithm 10 is the worst (0.0%).

The aggregate results for problem size 20×8×4 in Table B.11. For the evaluation criteria of producing solutions within 10% and within 5% of the best solution found thus far, all of the algorithms perform well (above 96%). The best diversification rule is to "never" diversify; this technique produces solutions within 1% of the optimal, 10.8% of the time. The 1-swap neighborhood is the best neighborhood function which produces the best solutions, getting within 1%, 14.6% of the time. The best parameter setting for γ is 0 (Simple Ascent) and produces the best results when the criteria requires solutions to be within 1% of optimal, achieving it 11.1% of the time.

4.2.4 Problem Size 20×14×4

The results for problem size 20×14×4 by algorithm are in Table B.12. For the evaluation criteria of producing solutions within 10% of the best solution found thus far, Algorithm 54 performed best (75.5%) and Algorithm 10 the worst (6.2%). For the evaluation criteria of producing solutions within 5% of the best solution found thus far, Algorithm 50 performed best (15.4%) and Algorithms 9, 10, and 11 the worst (0.0%). When the criteria required solutions to be with 1% of the optimal, all algorithms perform poorly (under 0.3%).

The aggregated results for problem size 20×14×4 in Table B.13 again show that the best diversification rule is to "never" diversify; this technique produces solutions within 10% of the optimal 49.1% of the time, and within 5%, 8.8% of the time. The 2-swap neighborhood is the preferred neighborhood function, producing solutions within 10%, 58.7% of the time. The 1-swap neighborhood is the best neighborhood function which produces the best solutions, getting within 1%, 10.3% of the time. The best parameter setting for γ is 0.9 (Simple Ascent) when the goal is within 10% and 5% of the best solution (47.7% and 7.2%). When the criteria required solutions to be with 1% of the optimal, all parameter settings perform poorly (under 0.2%).

4.2.5 Problem Size 30×12×4

The results for problem size 30×12×4 by algorithm are in Table B.14. These results are consistent with what the sensor saturated networks have already demonstrated thus far. For the evaluation criteria of producing solutions within 10% and within 5% of the best solution found thus far, all of the algorithms perform well (99.0% or better). When the criteria required solutions to be with 1% of the best solution found thus far, Algorithm 50 is the best (45.2%), and Algorithm 10 is the worst (0.0%).

The aggregate results for problem size 30×12×4 are in Table B.15. For the evaluation criteria of producing solutions within 10% and within 5% of the best solution found thus far, all of the algorithms perform well (above 99%). The best diversification rule is again to "never" diversify; this technique produces solutions within 1% of the optimal, 23.5% of the time. The 1-swap neighborhood is the best neighborhood function which produces the best solutions, getting within 1%, 32.4% of the time. The best parameter setting for γ is 0 (Simple Ascent) which produces the best results when the criteria requires solutions to be within 1% of the best solution found thus far, achieving it 18.3% of the time.

4.2.6 Problem Size 30×21×4

The results for problem size 30×21×4 by algorithm are in Table B.16. For the evaluation criteria of producing solutions within 10% of the best solution found thus far, Algorithm 50 performed best (75.1%) and Algorithm 10 the worst (0.2%). For the evaluation criteria of producing solutions within 5% of the best solution found thus far, Algorithm 50 performed best (20.9%) and Algorithms 6, 9, 10, 11, 12, 33, 34, 35, 36, 57, 58, 59, and 60 the worst (0.0%). When the criteria required solutions to be with 1% of the optimal, all algorithms perform poorly (under 0.2%).

The aggregate results for problem size 30×21×4 are in Table B.17. These results show that the best diversification rule is to "never" diversify; this technique

produces solutions within 10% of the best solution found thus far 44.8% of the time, and within 5%, 7.7% of the time. The 1-swap neighborhood is the neighborhood function which produces the best solutions: within 10%, 58.1% of the time and within 1%, 12.1% of the time. The best parameter setting for γ is 0.9 when the goal is within 10% of the best solution (38.0%) and a setting for γ of 0.7 when the goal is within 5% of the best solution (5.4%). When the criteria required solutions to be with 1% of the optimal, all parameter settings perform poorly (under 0.2%).

4.2.7 Problem Size 40×16×4

The results for problem size 40×16×4 by algorithm are in Table B.18. For the evaluation criteria of producing solutions within 10% and within 5% of the best solution found thus far, all of the algorithms perform well (99.5% or better). When the criteria required solutions to be with 1% of the best solution found thus far, Algorithm 49 is the best (68.3%), and Algorithms 11 and 12 perform the worst (0.0%).

The aggregate results for problem size 40×16×4 are in Table B.19. For the evaluation criteria of producing solutions within 10% and within 5% of the best solution found thus far, all of the algorithms perform well (above 99.9%). The best diversification rule is again to "never" diversify; this technique produces solutions within 1% of the optimal, 35.8% of the time. The 1-swap neighborhood is the neighborhood function which produces the best solutions (within 1%, 53.7% of the time). The best parameter setting for γ is 0 (Simple Ascent) which produces the best results when the criteria requires solutions to be within 1% of the best solution found thus far, achieving it 30.6% of the time.

4.2.8 Problem Size 40×28×4

The results for problem size 40×28×4 by algorithm are in Table B.20. For the evaluation criteria of producing solutions within 10% of the best solution found

thus far, Algorithm 50 performed best (76.6%) and Algorithms 9, 10, 11, and 12 are the worst (0.0%). For the evaluation criteria of producing solutions within 5% of the best solution found thus far, Algorithm 50 and 51 performed best (20.9%) and Algorithms 5, 6, 7, 8, 9, 10, 11, 12, 31, 32, 33, 34, 35, 36, 57, 58, 59, and 60 are the worst (0.0%). When the criteria required solutions to be with 1% of the optimal, all algorithms perform poorly (0.0%).

The aggregate results for problem size 40×28×4 are in Table B.21. These results show that the best diversification rule is to "never" diversify; this technique produces solutions within 10% of the best solution found thus far 41.8% of the time, and within 5%, 5.4% of the time. The 1-swap neighborhood is the neighborhood function which produces the best solutions: within 10%, 59.7% of the time and within 1%, 8.9% of the time. The best parameter setting for γ is 0.9 when the goal is within 10% of the best solution (38.0%) and a setting for γ of 0.8 when the goal is within 5% of the best solution (3.3%). When the criteria required solutions to be with 1% of the optimal, all parameter settings perform poorly (0.0%).

4.3 Overall Implementation Observations

4.3.1 Algorithm Type and Cooling Schedule

Simulated Annealing with a cooling schedule of 0.9 produces the highest quality results only in the sparse networks. In the sensor saturated networks, the Simple Ascent algorithm is more effective at returning high quality solutions. This may be because sensor saturated networks have far less local optimal than the sensor sparse networks.

4.3.2 Neighborhood Functions

The 1-swap and 2-swap neighborhoods proved to be the best neighborhoods by consistently performing well in the test problems. Larger neighborhoods performed

very poorly, especially in the larger problem sizes. In the larger neighborhoods, there are many different points to search with higher n-swap neighborhoods such that the probability of finding improving moves is smaller. In this case, the simpler methods are better.

4.3.3 Diversification Rules

The diversification rule which prevented diversification outperformed the diversification rules of switching to larger neighborhoods after 50% or 75% of the iterations. The implication for MUSAP is that starting off with a strategy and utilizing it throughout, the algorithm performs much better than attempting to switch midstream. This does not rule out the possibility of using a different type of diversification strategy that doesn't use the "switch after certain percentage of iterations" rule.

5. Conclusions and Future Research

5.1 Conclusion

This research has formulated the USSTRATCOM's sensor assignment problem as a type of resource allocation problem. It has further shown the utility of simulated annealing and simple ascent (iterative improvement) to solve that formulation. As the problems became more complex, simulated annealing with geometric cooling schedules emerged as the most effective algorithm.

5.2 Future Research

Future research could focus on four general areas. First, the independence assumptions could be relaxed, using a more robust objective function. Second, the specific parameters of the simulated annealing algorithm could be better specified. Third, the weights and priorities of the missions, along with mission drop thresholds could be added. And lastly, other heuristics techniques could be applied to this problem.

5.2.1 Inserting Dependant Probabilities

A separate research effort was underway concurrently with this research to determine the calculation of the objective function using dependant probabilities. Once that effort is complete, the algorithms in this research should be tested with the new objective function. As mentioned in Chapter 3, without the independence assumption, the number of feasible solutions that need to be searched increases exponentially. This implies that heuristics techniques will continue to be necessary to produce quality solutions in a reasonably short amount of time.

5.2.2 Simulated Annealing Parameter Specification

The precise ratio of inner loop to outer loop iterations was assumed by this research to be 10. That number should be examined to see if it creates the most efficient convergence behavior. The cooling schedule could also be examined with greater fidelity, as opposed to the few values that were examined. A dynamic search neighborhood could also be implemented in order to determine if more local optima can be avoided.

5.2.3 Varying Weights

This research assumed that all missions were weighed equally. Higher priority missions could be assigned to determine if the configurations provided give acceptable mission success probabilities. If those probabilities are not acceptable, a procedure for deciding the best missions to drop in order to maximize the remaining missions could be created.

5.2.4 Other Heuristics Techniques

Other techniques were not utilized primarily due to the desire for a fast, high quality solution. However, Genetic Algorithms (GA) have proven utility in the single objective versions of the series-parallel redundant assignment problem [41], so it is possible that GA could prove utility in this problem.

Bibliography

1. Aarts, E. and J. Korst. *Simulated Annealing and Bolzmann Machines*, Wiley, Chichester, 1989.
2. Aarts, E. and J. K. Lenstra. *Local Search in Combinatorial Optimization*, Princeton University Press, 2003.
3. Birolini, A. *Reliability Engineering: Theory and Practice*, Springer-Verlag, 2004.
4. Charnes, A. and W. W. Cooper. "The Theory of Search: Optimal Distribution of Effort". *Management Science* vol 5, 44-49, 1958.
5. Coit, D. W. and A. E. Smith. "Reliability Optimization of Series-Parallel Systems Using a Genetic Algorithm". *IEEE Transactions on Reliability* vol 45, 1996.
6. Coit, D. W. and A. E. Smith. "Penalty Guided Genetic Search for Reliability Design and Optimization". *Computers and Industrial Engineering* vol 30, 1996.
7. Collins, N. E., R. W. Eglese, and B. L. Golden. "Simmulated Annealing: an Annotated Bibliography". *American Journal of Mathematical and Management Sciences* vol 8, 1988.
8. Cook, S. A. *The Complexity of Theorem-Proving Procedures.* Proceedings of the 3rd Annual ACM Symposium on Theory of Computing Machinery, ACM, 151-158, 1971.
9. Dhingra, A. K. "Optimal Apportionment of Reliability and Redundancy in Series Systems Under Multiple Objectives". *IEEE Transactions on Reliability* vol 41, 1992.
10. Downsland K. A. *Simulated Annealing*, Modern Heuristic Techniques for Combinatorial Problems, Blackwell, Oxford, 1993.
11. Ehrgott, M. *Multicriteria Optimization*, Springer, 2005.
12. Elton, E. J., M. J. Gruber, and M. W. Padberg. "Simple Criteria for Optimal Portfolio Selection". *J. Finance* vol 31, 1341-1357, 1976.
13. Feo, T. A. and M. G. C. Resende. "A probabilistic heuristic for a computationally difficult set covering problem." Operations Research Letters, vol 8, 1989
14. Frederickson G. N. and D. B. Johnson. "The Complexity of selection and Ranking in X + Y and Matrices with Sorted Columns". *Journal on Computing Systems Science*, vol 24, 197-208, 1982.
15. Fyffe, D. E., W. W. Hines, N. K. Lee. "System Reliability Allocation and a Computational Algorithm". *IEEE Transactions on Reliability* vol 17, 1968.

16. Garey, M. and D. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, 1979.

17. Gen, M., K. Ida, M. Sasaki, and J. U. Lee. "Algorithm for Solving Large-scale 0-1 Goal Programming and its Application to Reliability Optimization Problem". *Computers and Industrial Engineering* vol 17, 1989.

18. Gen, M., K. Ida, Y. Tsuhimura, and C. E. Kim. "Large-scale 0-1 Fuzzy Goal Programming and its Application to Reliability Optimization Problem". *Computers and Industrial Engineering* vol 24, 1993.

19. Glover, F. and M. Laguna. *Tabu Search*, Kluwer Academic Publishers, 1997.

20. Haimes, Y., L. Lasdon, and D. Wismer. "On a bicriterion formulation of the problems of integrated system identification and system identification and systems optimization". *IEEE Transactions on Systems, Man, and Cybernetics* vol 1, 1971.

21. Hikita, M., Y. Nakagawa, K. Nakashima, and H. Narihisa. "Reliability Optimization of Systems by a Surrogate-constraints Algorithm". *IEEE Transactions on Reliability* vol 41, no. 3, 1992.

22. Holland, J. H. *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

23. Hsieh, Y. C. "A Linear Approximation for Redundant Reliability Problems with Multiple Component Choices". *Computers and Industrial Engineering* vol 44, 2002.

24. Ibaraki, T. and N. Katoh. *Resource Allocation Problems: Algorithmic Approaches*, MIT Press, New York, 1988.

25. Karp, R. M. "Reducibility among combinatorial problems". *Complexity of computer computations* (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972), 85-103, 1972

26. Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi. "Optimization by Simmulated Annealing". *Science* 220, 671-680, 1983

27. Koopman, B. O. "The Optimum Distribution of Effort". *J. Operations Res. Soc. Amer.* vol 1, 5263, 1953.

28. Kuo, W., and V. R. Prasad. "An Annotated Overview of System-Reliability Optimization". *IEEE Transactions on Reliability* vol 49, no. 2, 2000.

29. Ladner, R. E. "On the Structure of Polynomial Time Reducibility". *J. Assoc. Comput. Mach.* vol 22, 1975.

30. Li, D. and Y. Y. Haimes, "A Decomposition Method for Optimization of Large System Reliability". *IEEE Transactions on Reliability* vol 41, 1992.

31. Li, D. "Interactive Parametric Dynamic Programming and its Application in Reliability Optimization". *IEEE Transactions on Reliability* vol 191, 1995.

32. Luss, H. and S. K. Gupta. "Allocation of Effort Resources Among Competing Activities". *Operations Research* vol 5, 613-626, 1975.

33. Nakagawa, Y. and S. Miyazaki. "Surrogate Constraints Algorithm for Reliability Optimization Problems with Two Constraints". *IEEE Transactions on Reliability* vol 30, 1981.

34. Nocedal, J. and S. J. Wright. *Numerical Optimization*, Springer, 2006.

35. Powell, W. B. and B. Van Roy. "Approximate Dynamic Programming for High-Dimensional Resource Allocation Problems". Princeton University, 2003.

36. Sakawa, M. "Optimal Reliability Design of a Series-Parallel System by a Large-Scale Multi-Objective Optimization Method". *IEEE Transactions on Reliability* vol 30, 1981.

37. Singpurwalla, N. D. *Reliability and Risk: A Bayesian Perspective*, John Wiley and Sons, Inc., 2006.

38. Vidal, R. *Applied Simmulated Annealing*, Lecture Notes in Economics and Mathematical Systems. Springer, Berlin 1993.

39. Wolpert, D. H. and W. G. Macready. "No Free Lunch Theorems for Optimization". *IEEE Transactions on Evolutionary Computation* vol 1, no 1, 1997.

40. Wolsey, L. A. *Integer Programming*, John Wiley and Sons, Inc., 1998.

41. You, P. S. and T. C. Chen. "An Efficient Heuristic for Serier-Parallel Redundant Reliability Problems". *Computers and Operations Research* vol 32, 2005.

Appendix A. Experimental Settings

In order to test all possible combinations of the various factors that could effect algorithm performance, an experimental design with 72 different algorithm configurations was constructed. Those configurations are summarized in Tables A.1, A.2, and A.3.

Table A.1 Algorithm Configurations

ref	alg type	neighborhood	γ	starting point	diversify after
1	Simple Ascent	1-swap	0	marginal	50%
2	Simulated Annealing	1-swap	0.9	marginal	50%
3	Simulated Annealing	1-swap	0.8	marginal	50%
4	Simulated Annealing	1-swap	0.7	marginal	50%
5	Simple Ascent	2-swap	0	marginal	50%
6	Simulated Annealing	2-swap	0.9	marginal	50%
7	Simulated Annealing	2-swap	0.8	marginal	50%
8	Simulated Annealing	2-swap	0.7	marginal	50%
9	Simple Ascent	3-swap	0	marginal	50%
10	Simulated Annealing	3-swap	0.9	marginal	50%
11	Simulated Annealing	3-swap	0.8	marginal	50%
12	Simulated Annealing	3-swap	0.7	marginal	50%
13	Simple Ascent	1-swap	0	individual	50%
14	Simulated Annealing	1-swap	0.9	individual	50%
15	Simulated Annealing	1-swap	0.8	individual	50%
16	Simulated Annealing	1-swap	0.7	individual	50%
17	Simple Ascent	2-swap	0	individual	50%
18	Simulated Annealing	2-swap	0.9	individual	50%
19	Simulated Annealing	2-swap	0.8	individual	50%
20	Simulated Annealing	2-swap	0.7	individual	50%
21	Simple Ascent	3-swap	0	individual	50%
22	Simulated Annealing	3-swap	0.9	individual	50%
23	Simulated Annealing	3-swap	0.8	individual	50%
24	Simulated Annealing	3-swap	0.7	individual	50%

Table A.2 Algorithm Configurations (Con't)

ref	alg type	neighborhood	γ	starting point	diversify after
25	Simple Ascent	1-swap	0	marginal	75%
26	Simulated Annealing	1-swap	0.9	marginal	75%
27	Simulated Annealing	1-swap	0.8	marginal	75%
28	Simulated Annealing	1-swap	0.7	marginal	75%
29	Simple Ascent	2-swap	0	marginal	75%
30	Simulated Annealing	2-swap	0.9	marginal	75%
31	Simulated Annealing	2-swap	0.8	marginal	75%
32	Simulated Annealing	2-swap	0.7	marginal	75%
33	Simple Ascent	3-swap	0	marginal	75%
34	Simulated Annealing	3-swap	0.9	marginal	75%
35	Simulated Annealing	3-swap	0.8	marginal	75%
36	Simulated Annealing	3-swap	0.7	marginal	75%
37	Simple Ascent	1-swap	0	individual	75%
38	Simulated Annealing	1-swap	0.9	individual	75%
39	Simulated Annealing	1-swap	0.8	individual	75%
40	Simulated Annealing	1-swap	0.7	individual	75%
41	Simple Ascent	2-swap	0	individual	75%
42	Simulated Annealing	2-swap	0.9	individual	75%
43	Simulated Annealing	2-swap	0.8	individual	75%
44	Simulated Annealing	2-swap	0.7	individual	75%
45	Simple Ascent	3-swap	0	individual	75%
46	Simulated Annealing	3-swap	0.9	individual	75%
47	Simulated Annealing	3-swap	0.8	individual	75%
48	Simulated Annealing	3-swap	0.7	individual	75%

Table A.3 Algorithm Configurations (Con't)

ref	alg type	neighborhood	γ	starting point	diversify after
49	Simple Ascent	1-swap	0	marginal	never
50	Simulated Annealing	1-swap	0.9	marginal	never
51	Simulated Annealing	1-swap	0.8	marginal	never
52	Simulated Annealing	1-swap	0.7	marginal	never
53	Simple Ascent	2-swap	0	marginal	never
54	Simulated Annealing	2-swap	0.9	marginal	never
55	Simulated Annealing	2-swap	0.8	marginal	never
56	Simulated Annealing	2-swap	0.7	marginal	never
57	Simple Ascent	3-swap	0	marginal	never
58	Simulated Annealing	3-swap	0.9	marginal	never
59	Simulated Annealing	3-swap	0.8	marginal	never
60	Simulated Annealing	3-swap	0.7	marginal	never
61	Simple Ascent	1-swap	0	individual	never
62	Simulated Annealing	1-swap	0.9	individual	never
63	Simulated Annealing	1-swap	0.8	individual	never
64	Simulated Annealing	1-swap	0.7	individual	never
65	Simple Ascent	2-swap	0	individual	never
66	Simulated Annealing	2-swap	0.9	individual	never
67	Simulated Annealing	2-swap	0.8	individual	never
68	Simulated Annealing	2-swap	0.7	individual	never
69	Simple Ascent	3-swap	0	individual	never
70	Simulated Annealing	3-swap	0.9	individual	never
71	Simulated Annealing	3-swap	0.8	individual	never
72	Simulated Annealing	3-swap	0.7	individual	never

Appendix B. Results Tables

Table B.1 Optimal solutions for problem size 10×4×4 and 10×7×4

	Problem Instance									
size	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
10×4×4	3.88342	3.80867	3.71343	3.76942	3.73889	3.78874	3.73763	3.65259	3.73694	3.77561
10×7×4	5.59497	5.14499	5.11788	5.48895	5.64245	5.17348	5.79044	5.43078	5.32223	5.34225

Table B.2 Results for problem size 10×4×4

Algorithm												
dist	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
10%	84.4%	98.2%	94.4%	91.0%	89.2%	99.5%	97.4%	95.6%	95.0%	99.8%	98.6%	98.2%
5%	78.4%	88.0%	85.5%	84.3%	85.1%	90.4%	89.7%	88.0%	86.7%	84.6%	88.2%	86.7%
1%	16.6%	12.4%	13.0%	15.6%	15.9%	8.8%	12.1%	14.7%	9.9%	2.7%	6.5%	9.1%
Algorithm												
dist	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
10%	76.6%	98.4%	93.8%	92.3%	89.8%	99.9%	97.8%	95.7%	96.3%	99.5%	98.1%	98.4%
5%	74.7%	88.4%	85.5%	86.2%	86.9%	90.2%	90.8%	88.6%	92.5%	83.5%	88.0%	88.4%
1%	27.4%	13.4%	16.9%	19.5%	26.8%	7.9%	12.7%	16.4%	13.8%	3.5%	5.3%	8.2%
Algorithm												
dist	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**
10%	84.3%	98.5%	92.7%	90.6%	90.3%	99.5%	96.8%	95.5%	95.6%	99.4%	98.4%	97.5%
5%	78.8%	87.3%	82.9%	81.0%	85.7%	90.1%	89.0%	88.7%	90.2%	85.2%	89.2%	90.7%
1%	15.9%	12.1%	13.8%	12.6%	17.5%	10.8%	14.9%	17.8%	12.1%	4.0%	11.1%	10.1%
Algorithm												
dist	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**
10%	76.8%	97.7%	94.0%	93.5%	90.5%	99.2%	97.5%	95.7%	95.8%	99.5%	98.8%	97.7%
5%	75.5%	86.5%	84.8%	84.6%	88.2%	90.3%	89.9%	88.9%	91.6%	86.6%	91.4%	88.7%
1%	25.9%	9.8%	16.3%	15.0%	29.8%	12.2%	19.5%	17.9%	18.9%	3.8%	9.9%	8.8%
Algorithm												
dist	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**
10%	85.1%	97.6%	93.6%	91.0%	89.6%	99.2%	97.5%	94.9%	95.7%	99.8%	98.8%	97.3%
5%	76.5%	85.6%	84.2%	81.3%	84.6%	93.3%	91.5%	88.0%	91.8%	92.7%	92.5%	90.2%
1%	13.8%	14.2%	15.4%	13.6%	22.0%	15.9%	22.0%	23.4%	17.2%	9.0%	13.5%	16.9%
Algorithm												
dist	**61**	**62**	**63**	**64**	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**72**
10%	77.9%	97.9%	94.8%	93.0%	90.7%	99.5%	97.5%	94.7%	96.2%	99.5%	98.3%	98.0%
5%	75.6%	85.3%	83.1%	85.7%	88.7%	92.6%	89.5%	89.4%	93.2%	90.2%	91.0%	91.0%
1%	24.0%	13.8%	16.6%	18.2%	31.1%	15.4%	19.6%	22.6%	23.8%	6.5%	13.1%	15.7%

Table B.3 Aggregate results for problem size 10×4×4

Aggregate Results												
	Diversification Rule			**Neighborhood**			γ				**Construction**	
dist	**50%**	**75%**	**never**	**1-swp**	**2-swp**	**3-swp**	**0.9**	**0.8**	**0.7**	**0**	**Mrg**	**Ind**
10%	94.9%	94.8%	94.9%	91.2%	95.6%	97.9%	99.0%	96.6%	95.0%	88.9%	95.0%	94.8%
5%	86.6%	86.9%	87.8%	82.9%	89.1%	89.4%	88.4%	88.2%	87.2%	84.7%	86.9%	87.4%
1%	12.9%	14.2%	17.4%	16.1%	17.8%	10.6%	9.8%	14.0%	15.3%	20.1%	13.5%	16.1%

Table B.4 Results for problem size 10×7×4

	Algorithm											
dist	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
10%	9.0%	6.9%	1.8%	1.1%	9.6%	11.5%	5.2%	4.5%	8.4%	9.9%	6.8%	4.0%
5%	0.1%	0.2%	0.1%	0.0%	0.3%	1.1%	0.1%	0.0%	0.4%	0.4%	0.3%	0.3%
1%	0.0%	0.0%	0.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

	Algorithm											
dist	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
10%	6.8%	7.1%	2.1%	1.2%	13.5%	10.5%	5.6%	3.6%	19.7%	11.0%	4.9%	4.5%
5%	0.3%	0.3%	0.0%	0.0%	1.6%	0.3%	0.3%	0.2%	1.8%	0.2%	0.0%	0.2%
1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.1%	0.0%	0.0%	0.0%

	Algorithm											
dist	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**
10%	5.2%	6.2%	2.5%	0.9%	10.2%	12.3%	5.8%	4.4%	8.5%	12.3%	7.9%	5.0%
5%	0.0%	0.2%	0.0%	0.0%	0.2%	0.5%	0.4%	0.3%	0.1%	0.6%	0.6%	0.3%
1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

	Algorithm											
dist	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**
10%	5.5%	6.2%	2.0%	1.1%	13.1%	10.8%	5.2%	2.7%	22.1%	11.4%	8.4%	4.3%
5%	0.4%	0.2%	0.1%	0.0%	1.4%	0.4%	0.1%	0.2%	2.6%	0.7%	0.3%	0.1%
1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

	Algorithm											
dist	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**
10%	4.7%	5.3%	2.3%	1.1%	10.6%	12.1%	5.5%	4.1%	10.5%	11.9%	8.2%	5.8%
5%	0.0%	0.2%	0.1%	0.1%	0.1%	0.7%	0.1%	0.3%	0.4%	1.1%	0.5%	0.1%
1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

	Algorithm											
dist	**61**	**62**	**63**	**64**	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**72**
10%	4.7%	4.2%	1.8%	0.9%	12.8%	11.4%	6.2%	5.5%	21.0%	12.3%	8.4%	5.8%
5%	0.3%	0.4%	0.1%	0.0%	2.0%	0.7%	0.3%	0.3%	1.8%	0.8%	0.6%	0.4%
1%	0.0%	0.0%	0.0%	0.0%	0.1%	0.0%	0.0%	0.0%	0.1%	0.0%	0.0%	0.0%

Table B.5 Aggregate results for problem size 10×7×4

	Aggregate Results											
	Diversification Rule			**Neighborhood**			γ				**Construction**	
dist	**50%**	**75%**	**never**	**1-swp**	**2-swp**	**3-swp**	**0.9**	**0.8**	**0.7**	**0**	**Mrg**	**Ind**
10%	7.1%	7.3%	7.4%	3.8%	8.2%	9.7%	9.6%	5.0%	3.4%	10.9%	6.7%	7.7%
5%	0.4%	0.4%	0.5%	0.1%	0.5%	0.6%	0.5%	0.2%	0.2%	0.8%	0.3%	0.5%
1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

Table B.6 Results for problem size 10×4×4: Algorithm Modification

Algorithm												
dist	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
10%	99.3%	99.4%	99.5%	99.7%	99.9%	99.8%	99.8%	99.5%	100%	99.7%	99.7%	99.3%
5%	92.9%	89.5%	90.6%	91.6%	95.9%	90.6%	91.2%	92.0%	95.5%	84.5%	90.1%	88.5%
1%	17.6%	13.4%	17.2%	17.6%	19.0%	8.3%	13.4%	16.5%	8.3%	2.2%	6.1%	6.8%
Algorithm												
dist	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**
10%	99.3%	99.3%	98.8%	99.5%	100%	100%	99.8%	99.9%	100%	99.9%	99.9%	99.6%
5%	90.4%	90.2%	88.0%	88.2%	95.9%	92.2%	92.0%	94.6%	95.5%	89.4%	90.1%	92.6%
1%	16.2%	11.1%	16.2%	16.3%	21.5%	11.5%	18.4%	20.8%	12.7%	3.2%	11.1%	11.8%
Algorithm												
dist	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**
10%	99.0%	98.8%	99.5%	99.5%	100%	99.7%	99.8%	100%	100%	99.8%	99.9%	99.9%
5%	91.7%	87.1%	86.5%	89.8%	95.8%	93.3%	93.6%	94.6%	96.8%	90.8%	92.0%	94.6%
1%	17.5%	13.3%	14.9%	16.7%	25.9%	17.7%	22.3%	22.6%	16.7%	8.8%	13.0%	15.8%

Table B.7 Aggregate results for problem size 10×4×4: Algorithm Modification

Aggregate Results										
	Diversification Rule			**Neighborhood**			γ			
dist	**50%**	**75%**	**never**	**1-swp**	**2-swp**	**3-swp**	**0.9**	**0.8**	**0.7**	**0**
10%	99.6%	99.7%	99.7%	99.3%	99.9%	99.8%	99.6%	99.6%	99.7%	99.7%
5%	91.1%	91.6%	92.2%	89.7%	93.5%	91.7%	89.7%	90.5%	91.8%	94.5%
1%	12.2%	14.2%	17.1%	15.7%	18.2%	9.7%	9.9%	14.7%	16.1%	17.3%

Table B.8 Results for problem size 10×7×4: Algorithm Modification

Algorithm												
dist	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
10%	50.8%	65.3%	54.4%	53.2%	67.5%	71.9%	70.0%	66.7%	58.5%	53.2%	55.4%	58.7%
5%	22.2%	25.1%	19.4%	20.5%	28.1%	30.2%	26.7%	27.9%	18.3%	11.7%	12.2%	15.4%
1%	0.7%	1.0%	0.3%	0.6%	1.5%	0.6%	0.6%	0.9%	0.2%	0.0%	0.1%	0.1%
Algorithm												
dist	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**
10%	41.0%	55.0%	49.3%	43.1%	68.6%	73.0%	66.9%	72.3%	62.9%	61.9%	64.0%	64.4%
5%	13.0%	17.2%	16.2%	12.3%	29.7%	33.6%	28.8%	32.4%	21.3%	17.3%	18.7%	19.4%
1%	0.7%	0.7%	0.5%	0.1%	1.9%	0.8%	1.2%	2.3%	0.5%	0.2%	0.4%	0.2%
Algorithm												
dist	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**
10%	33.7%	48.3%	41.2%	38.7%	71.3%	76.3%	74.5%	73.0%	68.1%	70.3%	70.5%	69.7%
5%	10.2%	15.0%	12.1%	12.9%	34.0%	35.0%	34.7%	33.2%	25.4%	25.6%	28.1%	26.0%
1%	0.4%	0.6%	0.4%	0.4%	2.4%	1.8%	2.5%	1.7%	1.4%	0.9%	0.5%	1.0%

Table B.9 Aggregate results for problem size 10×7×4: Algorithm Modification

Aggregate Results										
	Diversification Rule			**Neighborhood**			γ			
dist	**50%**	**75%**	**never**	**1-swp**	**2-swp**	**3-swp**	**0.9**	**0.8**	**0.7**	**0**
10%	60.5%	60.2%	61.3%	47.8%	71.0%	63.1%	63.9%	60.7%	60.0%	58.0%
5%	21.5%	21.7%	24.4%	16.3%	31.2%	20.0%	23.4%	21.9%	22.2%	22.5%
1%	0.6%	0.8%	1.2%	0.5%	1.5%	0.5%	0.7%	0.7%	0.8%	1.1%

Table B.10 Results for problem size 20×8×4: Algorithm Modification

Algorithm												
dist	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
10%	99.2%	100%	100%	99.9%	99.7%	100%	100%	100%	99.9%	100%	100%	100%
5%	98.0%	99.5%	99.5%	99.4%	98.9%	98.8%	98.8%	98.2%	95.2%	94.2%	94.3%	95.0%
1%	17.7%	10.1%	9.0%	10.3%	8.2%	2.2%	1.8%	2.2%	0.2%	0.0%	0.1%	0.3%
Algorithm												
dist	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**
10%	98.8%	100%	99.9%	99.5%	99.9%	100%	99.9%	100%	100%	100%	100%	100%
5%	97.7%	99.6%	99.1%	98.7%	98.9%	99.6%	99.1%	99.3%	97.8%	96.7%	97.1%	96.1%
1%	17.7%	12.8%	12.5%	14.1%	12.8%	4.3%	4.7%	6.0%	2.0%	0.2%	0.4%	0.6%
Algorithm												
dist	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**
10%	96.8%	100%	99.8%	99.9%	99.5%	100%	99.9%	100%	99.8%	100%	100%	100%
5%	96.0%	99.4%	99.3%	99.2%	99.1%	99.8%	99.5%	99.6%	99.0%	98.5%	98.7%	98.7%
1%	19.8%	17.0%	16.4%	17.4%	19.3%	10.7%	10.1%	12.4%	2.5%	1.1%	1.3%	1.3%

Table B.11 Aggregate results for problem size 20×8×4: Algorithm Modification

Aggregate Results										
	Diversification Rule			**Neighborhood**			γ			
dist	**50%**	**75%**	**never**	**1-swp**	**2-swp**	**3-swp**	**0.9**	**0.8**	**0.7**	**0**
10%	99.9%	99.8%	99.6%	99.5%	99.9%	100%	100%	99.9%	99.9%	99.3%
5%	97.5%	98.3%	98.9%	98.8%	99.1%	96.8%	98.5%	98.4%	98.2%	97.8%
1%	5.2%	7.3%	10.8%	14.6%	7.9%	0.8%	6.5%	6.3%	7.2%	11.1%

Table B.12 Results for problem size 20×14×4: Algorithm Modification

Algorithm												
dist	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
10%	30.7%	66.2%	54.4%	52.6%	43.5%	47.1%	46.0%	46.5%	10.7%	6.2%	7.6%	8.4%
5%	6.2%	13.3%	10.4%	10.1%	3.4%	1.1%	2.1%	1.8%	0.0%	0.0%	0.0%	0.1%
1%	0.1%	0.0%	0.2%	0.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Algorithm												
dist	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**
10%	27.4%	63.0%	50.0%	49.9%	54.6%	63.9%	65.3%	62.4%	18.9%	15.0%	17.4%	17.8%
5%	7.6%	12.7%	9.0%	9.9%	8.8%	6.9%	6.9%	9.2%	0.1%	0.0%	0.2%	0.1%
1%	0.0%	0.0%	0.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Algorithm												
dist	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**
10%	26.4%	62.3%	54.4%	48.1%	58.4%	75.5%	71.1%	69.6%	29.6%	29.9%	30.8%	32.7%
5%	7.7%	15.4%	10.6%	10.4%	13.7%	14.6%	14.4%	15.3%	1.0%	1.0%	0.7%	0.9%
1%	0.0%	0.1%	0.0%	0.0%	0.0%	0.0%	0.1%	0.0%	0.0%	0.0%	0.0%	0.0%

Table B.13 Aggregate results for problem size 20×14×4: Algorithm Modification

Aggregate Results										
	Diversification Rule			**Neighborhood**			γ			
dist	**50%**	**75%**	**never**	**1-swp**	**2-swp**	**3-swp**	**0.9**	**0.8**	**0.7**	**0**
10%	35.0%	42.1%	49.1%	48.8%	58.7%	18.8%	47.7%	44.1%	43.1%	33.4%
5%	4.0%	6.0%	8.8%	10.3%	8.2%	0.3%	7.2%	6.0%	6.4%	5.4%
1%	0.0%	0.0%	0.0%	0.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

Table B.14 Results for problem size 30×12×4: Algorithm Modification

Algorithm												
dist	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
10%	99.9%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5%	99.6%	100%	100%	99.9%	99.8%	100%	100%	100%	99.9%	99.9%	99.9%	99.7%
1%	27.4%	24.3%	19.7%	22.4%	7.4%	4.9%	4.6%	7.9%	0.3%	0.0%	0.2%	0.2%
Algorithm												
dist	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**
10%	99.9%	100%	99.9%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5%	99.0%	100%	99.9%	99.7%	99.8%	100%	100%	100%	100%	100%	100%	100%
1%	38.2%	34.5%	29.6%	30.9%	14.2%	14.9%	14.4%	13.9%	1.2%	0.5%	0.7%	0.8%
Algorithm												
dist	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**
10%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5%	99.2%	100%	99.9%	99.6%	100%	100%	100%	100%	100%	100%	100%	100%
1%	41.5%	45.2%	36.9%	38.6%	32.1%	26.7%	26.4%	23.8%	2.7%	2.7%	2.4%	3.5%

Table B.15 Aggregate results for problem size 30×12×4: Algorithm Modification

Aggregate Results										
	Diversification Rule			**Neighborhood**			γ			
dist	**50%**	**75%**	**never**	**1-swp**	**2-swp**	**3-swp**	**0.9**	**0.8**	**0.7**	**0**
10%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5%	99.9%	99.9%	99.9%	99.7%	100%	100%	100%	100%	99.9%	99.7%
1%	9.9%	16.2%	23.5%	32.4%	15.9%	1.3%	17.1%	15.0%	15.8%	18.3%

Table B.16 Results for problem size 30×21×4: Algorithm Modification

Algorithm												
dist	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
10%	39.0%	60.6%	56.2%	53.7%	25.5%	19.2%	27.3%	26.3%	0.7%	0.2%	0.2%	0.4%
5%	5.3%	7.2%	7.3%	6.7%	0.4%	0.0%	0.1%	0.1%	0.0%	0.0%	0.0%	0.0%
1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Algorithm												
dist	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**
10%	43.4%	69.1%	61.9%	61.5%	42.8%	45.4%	49.6%	48.2%	2.3%	1.4%	2.1%	2.3%
5%	10.0%	14.3%	13.0%	12.5%	1.4%	1.1%	0.7%	2.3%	0.0%	0.0%	0.0%	0.0%
1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Algorithm												
dist	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**
10%	44.6%	75.1%	65.2%	66.4%	58.7%	65.2%	65.7%	66.4%	8.6%	5.8%	7.3%	9.0%
5%	11.1%	20.9%	17.9%	19.1%	5.5%	4.3%	5.6%	8.3%	0.0%	0.0%	0.0%	0.0%
1%	0.1%	0.0%	0.0%	0.5%	0.0%	0.0%	0.0%	0.1%	0.0%	0.0%	0.0%	0.0%

Table B.17 Aggregate results for problem size 30×21×4: Algorithm Modification

Aggregate Results										
	Diversification Rule			**Neighborhood**			γ			
dist	**50%**	**75%**	**never**	**1-swp**	**2-swp**	**3-swp**	**0.9**	**0.8**	**0.7**	**0**
10%	25.8%	35.8%	44.8%	58.1%	45.0%	3.4%	38.0%	37.3%	37.1%	29.5%
5%	2.3%	4.6%	7.7%	12.1%	2.5%	0.0%	5.3%	5.0%	5.4%	3.7%
1%	0.0%	0.0%	0.1%	0.1%	0.0%	0.0%	0.0%	0.0%	0.1%	0.0%

Table B.18 Results for problem size 40×16×4: Algorithm Modification

	Algorithm											
dist	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
10%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5%	100%	100%	99.7%	99.7%	99.9%	100%	100%	100%	100%	100%	100%	100%
1%	50.5%	38.6%	36.9%	39.9%	12.4%	7.7%	9.4%	7.5%	0.7%	0.1%	0.0%	0.0%

	Algorithm											
dist	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**
10%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5%	100%	100%	99.9%	99.6%	100%	100%	100%	100%	100%	100%	100%	100%
1%	62.0%	56.3%	51.0%	52.6%	28.8%	23.3%	19.1%	21.8%	1.6%	0.8%	1.0%	1.0%

	Algorithm											
dist	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**
10%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5%	100%	100%	99.9%	99.6%	99.9%	100%	100%	100%	100%	100%	100%	100%
1%	68.3%	64.9%	63.5%	59.7%	46.7%	41.9%	36.8%	36.4%	4.6%	2.4%	2.4%	1.8%

Table B.19 Aggregate results for problem size 40×16×4: Algorithm Modification

	Aggregate Results									
	Diversification Rule			**Neighborhood**			γ			
dist	**50%**	**75%**	**never**	**1-swp**	**2-swp**	**3-swp**	**0.9**	**0.8**	**0.7**	**0**
10%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5%	99.9%	100%	100%	99.9%	100%	100%	100%	99.9%	99.9%	100%
1%	17.0%	26.6%	35.8%	53.7%	24.3%	1.4%	26.2%	24.5%	24.5%	30.6%

Table B.20 Results for problem size 40×28×4: Algorithm Modification

	Algorithm											
dist	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
10%	40.3%	52.0%	48.0%	49.1%	13.0%	5.6%	10.2%	12.9%	0.0%	0.0%	0.0%	0.0%
5%	2.7%	2.2%	3.0%	3.3%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

	Algorithm											
dist	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**
10%	53.7%	66.7%	60.6%	61.3%	31.9%	25.6%	28.9%	30.0%	0.2%	0.1%	0.1%	0.3%
5%	4.5%	7.8%	8.7%	7.4%	0.6%	0.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

	Algorithm											
dist	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**
10%	63.9%	76.6%	70.8%	73.0%	54.1%	51.1%	52.1%	55.1%	2.3%	1.3%	0.3%	1.2%
5%	9.6%	16.9%	16.9%	15.5%	2.5%	0.8%	1.2%	1.7%	0.0%	0.0%	0.0%	0.0%
1%	0.0%	0.0%	0.0%	0.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

Table B.21 Aggregate results for problem size 40×28×4: Algorithm Modification

	Aggregate Results									
	Diversification Rule			**Neighborhood**			γ			
dist	**50%**	**75%**	**never**	**1-swp**	**2-swp**	**3-swp**	**0.9**	**0.8**	**0.7**	**0**
10%	19.3%	30.0%	41.8%	59.7%	30.9%	0.5%	31.0%	30.1%	31.4%	28.8%
5%	0.9%	2.4%	5.4%	8.2%	0.6%	0.0%	3.1%	3.3%	3.1%	2.2%
1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
27–03–2008	Master's Thesis	Mar 2007 — Mar 2008

4. TITLE AND SUBTITLE

INTELLIGENCE SURVEILLANCE AND RECONNAISSANCE ASSET ASSIGNMENT FOR OPTIMAL MISSION EFFECTIVENESS

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Kappedal, Ryan D., Capt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GOR/ENC/08-10

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

USSTRATCOM/J811
Attn: Maj David Pugh
OFFUTT AFB, NE 68113-6500

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This research develops mathematical programming techniques to solve an intelligence, surveillance, and reconnaissance sensor assignment problem for USSTRATCOM. The problem as specified is hypothesized to be difficult (i.e. NP-HARD). With the smallest test cases, the true optimal solution is found using simple optimization techniques, but, due to intractability, the optimal solutions for larger test cases are not found using these same techniques. Instead, heuristic techniques are applied to several test cases in order to determine the best, robust methodologies to find true or near optimal solutions. Specifically, simulated annealing (SA) is tested for convergence properties across several different parameter settings. This research also utilizes local search techniques with simple exchange neighborhoods of various sizes. Mission prioritization is also examined via a weighted sum scalarization technique.

15. SUBJECT TERMS

Intelligence, Surveillance, Reconnaissance, Simulated Annealing, Heuristic, Integer Programming

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Maj. August G. Roesener, Phd, (ENS)
U	U	U	UU	78	19b. TELEPHONE NUMBER *(include area code)* (937) 785-3636; e-mail: august.roesener@afit.edu